A Scheduling Mechanism Approach to Web Servers based on Processes' Behavior for Software Evolution

Sukanya Suranauwarat, Taniguchi Hideo and Ushijima Kazuo

Graduate School of Information Science and Electrical Engineering, Kyushu University

1 Introduction

When considering the total costs of a software system over its lifetime, it turns out that on average maintenance alone consumes 50-75% of these costs [1, 2]. The IEEE estimates that companies in the United States spend more than $70 billion annually to maintain existing software [3]. Therefore, software maintenance is an area where even small improvements to the process can potentially reap significant benefits [4]. The real maintenance activity, that is the correction of faults, accounts for only about 25% of the total maintenance effort. Approximately another 25% of the maintenance effort concerns adapting software to environmental changes such as new hardware, while half of the maintenance cost is spent on changes to accommodate changing user requirements such as extra functions to be provided by the system, or increasing the system's performance [5]. Changes in both the system's environment and user requirements are inevitable. Software models part of reality, and reality changes. So the software has to change too. It has to evolve. A large percentage of what we are used to calling maintenance is actually software evolution.

The following idea could be a way to reduce the maintenance cost due to software evolution. Our idea is that by increasing operating system abilities, operating system could adapt existing software to new or changed user requirements without making any changes to the existing software. Our idea requires that the operating system have abilities to observe the software's execution behavior and evolve the software' s execution behavior based on observed results. By using these abilities, the operating system could optimize software's execution behavior allowing user requirements to be satisfied without making any changes to the existing software and hardware.

We have already applied this idea to a Web server and the user requirement given consideration was improving response time when the processor of the server machine became bottlenecked. And We evaluated the performance of a Web server in simple cases, such as when the number of browsers accessing the Web server ranged from 1 to 3 and just two machines were used. Our experimental results showed that the response time was improved greatly when the processor of the server machine became bottlenecked and the parameters set or predicted by our mechanism based on the Web server's execution behavior were used [6]. However, the bottleneck of the processor in [6] was caused by a coexisting processor-bound process which we used for simulating the situation that the processor of the server machine is bottlenecked.

In this paper, we give an overview of how we observe and alter a Web server's execution behavior, evaluate the performance of a Web server when it is accessed by a lot of browsers at the same time which is more likely to be a realistic situation and could be the situation that causes the processor to become bottlenecked, then compare that with [6] especially when the processor-bound process does not coexist, and verify its effectiveness.

2 Overview

We observe the Web server's execution behavior through the logging mechanism and alter its execution behavior by using the process control mechanism. Both mechanisms are integrated into the operating system.

2.1 logging mechanism

When a Web server is running, a log is collected recording the information necessary to determine the optimal execution. A log is a sequence of entries describing process identifier, process state and time. Then a sequence called PFS (Program Flow Sequence) is created for each process. Note that a Web server is normally composed of multiple processes. PFS is a sequence of entries describing process state and time spent.

2.2 process control mechanism

For this paper, the content of a Web page is pretty basic and simple, which is only composed of 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 Web page, the browser will interpret and process the HTML file sent back by the Web server it requested and then display the text data. During the interpretation, if the Web page is also composed of image data, then the browser will request the Web server again for the Image file. When a Web site becomes busy (i.e., a Web server is accessed by a lot of browsers), 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 popular Web sites. So, while a Web site is busy, we thought that it is much better if the text data shows up on any browsers relatively faster. Therefore, we considered improving the time from requesting a Web 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. And then 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 at a ready queue or for the resource needed for an event to become available at a sleep queue, we can achieve enhanced response time. According to this, we proposed the policies that when a processor (a hard disk or a network communication) becomes bottlenecked, any server process handling a HTML file will be moved to the head of the ready queue (sleep queue associated with the event) [6].

When we discussed the process control mechanism that implements the above policy focused on the bottleneck of the processor [6], 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 behavior of a Web server. We analyzed the Web server's processes behavior based on PFS and found out that any server process handling a HTML file has 2 characteristics: runs after waiting for a long time in the wait state (characteristic 1) and tends to cycle between run state and wait state fewer times than that of 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 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 an server process handling a HTML file. By these parameters, we can detect which process appears to be a server process handling a 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 why processes that loose characteristic 2 are moved to the back of the ready queue is that sometimes server processes handling Image files are mistaken as server processes handling HTML files because they exhibit characteristic 1.

Our process control mechanism predicts and updates SLP and RW based on PFS automatically every 50 milliseconds. How to predict SLP and RW is described in [6]

3 Performance

In this section, we present experiments designed to evaluate the effectiveness of the process control mechanism we designed. We start with a description of the experimental setup, and proceed to present the results of various experiments.

3.1 Experimental Setup

The software used for the Web server and the browser in our experiment was Apache version 1.2.5 and Netscape Navigator version 3.04 respectively. The Web server ran on the 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/UNIX version 2.1 in single user mode and were connected by a private 10Mb/s Ethernet. During the experiment, the operating system 's I/O buffer cache in the server machine and each browser's cache were disabled.

The Web server was accessed by three browsers from each of the three machines at the same time. 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 Web page until text data starts displaying and the time (t2) from requesting a Web 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 Web server simultaneously every 30 seconds when the Web server coexisted with a processor-bound process and SLP was fixed at 20 seconds by our process control mechanism without using PFS. The purpose for this experiment is to know how the response time of text data would be improved in the situation that is the best for our process control mechanism. In experiment 2, all the browsers accessed the Web server randomly without any processor-bound process and SLP was set in the same way as in experiment 1, while in experiment 3, SLP was predicted and updated automatically based on PFS by our control mechanism every 50 milliseconds. The purpose for experiment 2 and 3 are to know how the response time of text data would be improved when SLP was set not based on and based on the Web server's execution behavior respectively in the more realistic situation.

3.2 Experimental Result

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

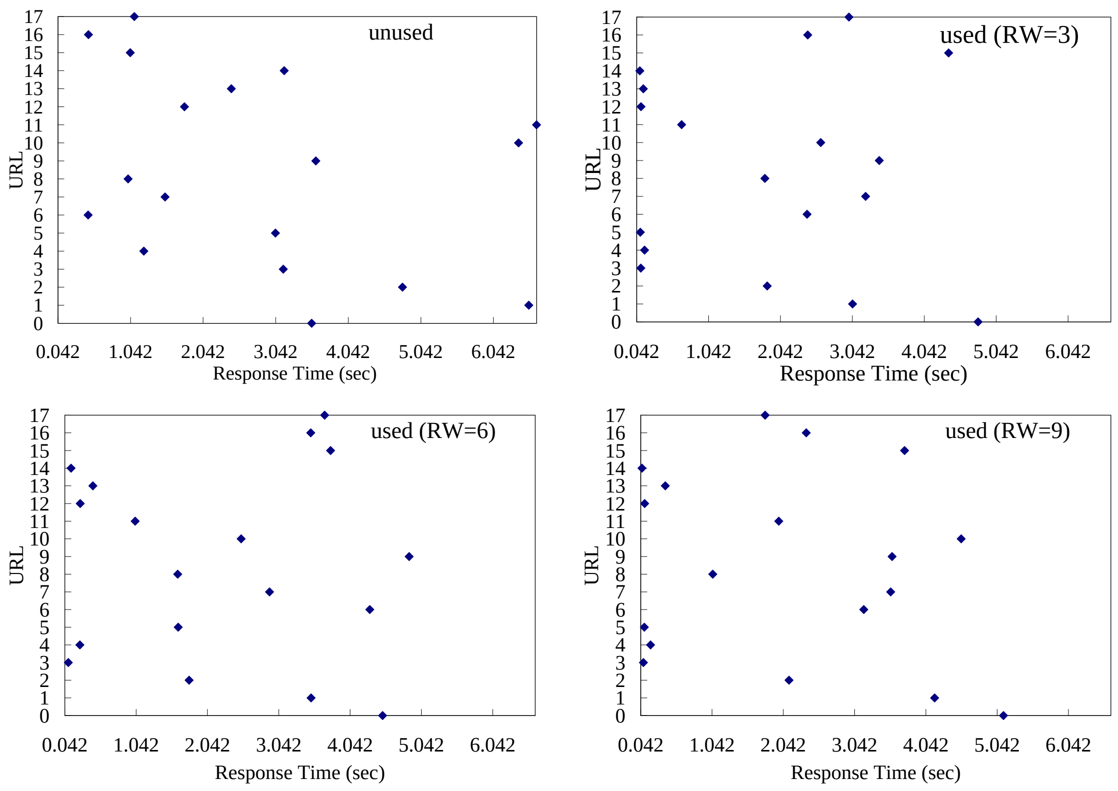

Figure 1 Response Time of Text Data in Experiment 1

Figure 2(a) illustrates the mean and the range of response times of text data from Figure 1 into one graph and shows that the mean response time of text data when using our process control mechanism is faster than when not using it. This case produced the best improvement of the response time of text data for this experiment and in the past [6], because the coexisting processor-bound process always caused the processor to become bottlenecked and the server processes waiting for HTML files requests from browsers were always in the wait state at least 30 seconds which was more than SLP (20 seconds).

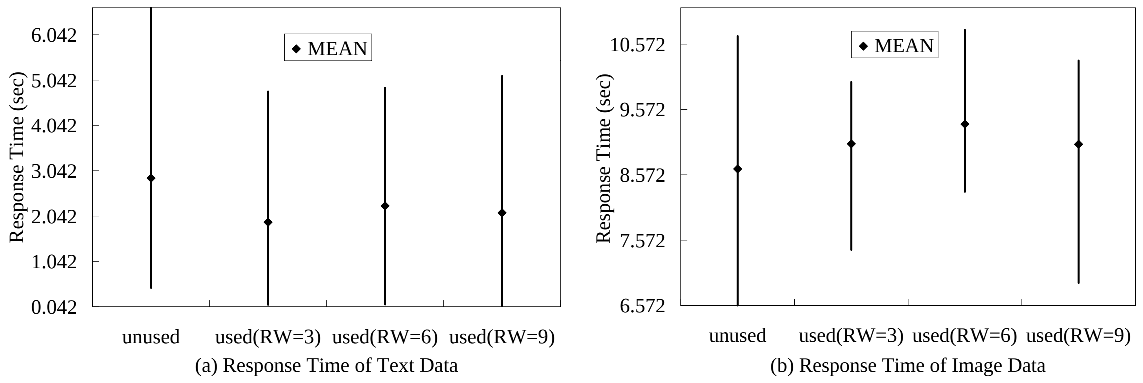

Figure 2 The Effect of Our Process Control Mechanism in Experiment 1

The rest of the experimental results will be shown like figure 2(a).

Figure 2(b) illustrates the response time 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 process control mechanism. 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.

In experiment 2 and 3, the results were desirable. Unfortunately, there is not sufficient space to discuss the results in this extended abstract. So, if this extended abstract gets accepted, we will also discuss the rest of the experimental results in the full paper.

The experimental results show that a lot of accesses from browsers at the same time can cause the processor of the server machine to become bottlenecked. Also, the parameter SLP is effectively predicted and updated based on the Web server's execution behavior by our mechanism, and is comparable to experiment 1 which for our process control mechanism produces the best improvement.

4 Conclusion

The total cost of system maintenance is estimated to comprise at least 50% of total life cycle costs. Half of the maintenance cost is spent on changes to accommodate changing user requirements. The following idea could be a way to reduce the maintenance cost due to adapting the software to new or changed user requirements. Our idea is that by increasing operating system abilities to observe the software's execution behavior and alter the software's execution behavior based on observed results, the operating system could optimize software execution behavior allowing user requirements to be satisfied without making any changes to the existing software and hardware. We have already applied this idea to a Web server and evaluated the performance of a Web server in simple cases [6] and the user requirement given consideration was improving response time when the processor of the server machine became bottlenecked. In this paper, we evaluated the performance of a Web server when it is busy which is a more realistic situation. The experimental results showed that the performance of the Web server was improved greatly. This means that a lot of accesses from browsers at the same time can cause the processor of the server to become bottlenecked and the parameter we used to determine the optimal execution of the Web server was effectively predicted and updated based on the Web server's execution behavior by our mechanism.

References

[1] B.W. Boehm, Software engineering, IEEE Transactions on Computer C-25, pp.1226-1241,December 1976.

[2] B.P. Lientz and E.B. Swanson, Software Maintenance Maintenance Management, Addison-Wesley, 1980.

[3] Sutherland, J., “Business Objects in Corporate Information Systems”, ACM Computing Surveys, 27(2), pp.274-276, June 1995.

[4] Chris F. Kemerer, A Longitudinal Empirical Analysis of Software Evolution, International Workshop on Principles of Software Evolution, pp. 23-28, April 1998.

[5] Hans Van Vliet, Software engineering:Principles and Practice, John Wiley & Sons Ltd, 1992.

[6] Sukanya Suranauwarat and Hideo Taniguchi, Process Scheduling Policy for a WWW Server Based on its Contents, Trans. of IPSJ Vol.40, No.6 (to appear) (In Japanese).