2L−6

Evaluation of a Process' Behavior-based Disk Scheduling Policy for a WWW Server

Sukanya Suranauwarat Hideo Taniguchi
Graduate School of Information Science and Electrical Engineering, Kyushu University

1 Introduction

As the demand placed on WWW (World Wide Web) servers grows, the number of simultaneous requests they must handle increases. As a result, users see slower response times during periods of high demand of WWW requests. In other words, it takes a longer time for the first data to display on browsers, the text data stored in an HTML (HyperText Markup Language) file, to start displaying when servers are accessed by many requests simultaneously. This situation could be one in which it is most desirable to improve the response time and this can be achieved by scheduling resources more efficiently in operating systems. We have proposed a disk scheduling policy for improving response time of a WWW server[1]. This policy gives preferential use of the disk drive to any process that is predicted based on its behavior to be a server process handling an HTML file request, by moving its I/O requests to the head of the I/O queue. In this paper, we present the experimental evaluation of our proposed disk scheduling policy with regard to the number of simultaneous requests from browsers.

2 Overview

This section gives a brief overview of *how our disk scheduler detects which processes are server processes handling HTML file requests* and *how our disk scheduler operates the I/O queue*, in order to provide sufficient understanding to the rest of the paper. A more detail about these can be found in [1].

2.1 How To Detect The Object Processes

We first need to know what a server process handling an HTML file request is like. So, we analyzed the execution behavior of a WWW server and found that any server process handling an HTML file request 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 other types of file request (characteristic 2).

Next, we introduced two parameters into our scheduler in order to determine which processes have the above two characteristics: 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 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 request. By these two parameters, we can detect which process appears to be a server process handling an HTML file request. SLP and RW are automatically predicted and updated every time period based on the execution behavior information of each server process called *PFS (Program Flow Sequence)*. PFS is a sequence of entries describing process state and time spent. PFS of each server process is created and updated every time period based on the log we collected when the WWW server is running. A log is a sequence of entries describing process identifier, process state and time.

2.2 How To Operate The I/O Queue

Our scheduler puts any I/O request from any process that has characteristic 1 at the head of the I/O queue; and when that process loses characteristic 2, its I/O requests will be scheduled normally, i.e., its I/O requests will be put into the I/O queue using a routine provided by the operating system, which in our case is the *disksort* routine. Disksort enters I/O requests into the queue in a cyclic, ascending, cylinder order.

3 Experimental Evaluation

3.1 Experimental Setup

Our disk scheduler is implemented in BSD/OS 2.1. The server and client software used in our experiment were Apache 1.2.5. and Netscape Navigator 3.04 respectively. 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. All the machines were connected by a private 10 Mb/s Ethernet. Also, the experiment was 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 experiment, in order to clearly see the effect of our disk scheduler.

In our experiment, we varied the number of client machines from 1 to 4, and from each machine we set three browsers to access the WWW server randomly at the same time. All browsers accessed unique URLs

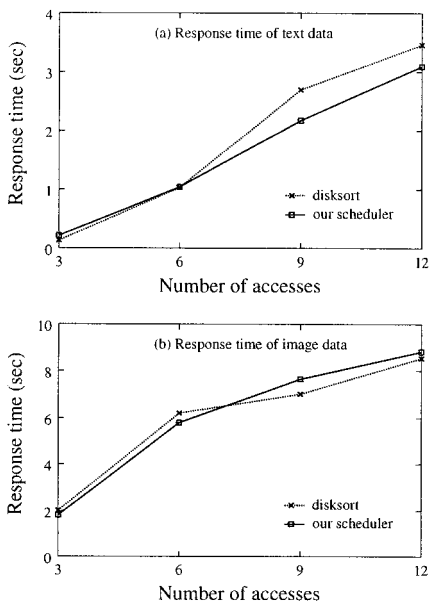

Fig.1. The experimental results when the WWW server is accessed by multiple browsers randomly at the same time.

(URL – Uniform Resource Locator) all of which have the same content, an HTML file (1,772 bytes) and an Image file (43,770 bytes). For each URL, we measured the 5 trial times of *time1* and *time2*. 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 trial times of time1 and time2 as response time of text data and response time of image data respectively. During the experiment, SLP and RW were automatically predicted and updated based on PFS every 500 milliseconds. And our previous work[2] has already showed that SLP and RW are effectively predicted and updated by our scheduler.

3.2 Experimental Results

Figure 1 shows the mean response times of text data and image data from all URLs when using our disk scheduler and when using disksort.

In Fig. 1(a), we did not notice the effect of our disk scheduler when the number of accesses is small (i.e., less than 6). This could be because our scheduler only moves a server process handling an HTML file request to the head of the I/O queue when the disk drive is bottlenecked, and this barely occurs when the number of accesses is small. For example, when the number of accesses is 3, the percentage of disk bottleneck is only 3% according to the content of the I/O queue we logged during the experiment. On the other hand, Figure 1(a) shows that when the number of accesses increases (i.e., more than 6), our disk scheduler produces a good improvement. To be more specific, the response times of text data when the number of accesses is 9 and 12 are improved 19% and 11% respectively. These figures are calculated by $\frac{a-b}{a} \times 100\%$ where a and b are the mean response times of text data when using disksort and when using our scheduler respectively. The good improvements when the number of accesses is 9 and 12 agree with their high values in the percentage of disk bottleneck, which is about 6 times of that when the number of accesses is 3, according to the content of the I/O queue we logged during the experiment.

However, care must be taken to ensure that the resulting unfairness due to our policy of giving favorable treatment to sever processes handling HTML file requests, does not outweigh the performance gains obtained. In our experiment, as shown in Fig. 1(b), the mean response times of image data when using our scheduler is not so different from when using disksort. Accordingly, response time of image data pays a small penalty under our scheduler.

Therefore, any WWW server that experiences a lot of simultaneous accesses from users would benefit from our disk scheduler.

4 Conclusion

This paper examined the performance of a WWW server with regard to the number of simultaneous requests from browsers, when using the proposed disk scheduler. And our experimental result when the WWW server is accessed randomly by multiple requests at the same time shows that by using our disk scheduler the response time of text data, the time from requesting a WWW page until text data starts displaying, can be reduced. Moreover, the effect of unfairness due to our policy of giving favorable treatment to sever processes handling HTML file requests on the response times of other types of data, which in our case is image data, is relatively small. Therefore, any WWW server that experiences a lot of simultaneous requests from users would benefit from our disk scheduler.

References

[1] S. Suranauwarat and H. Taniguchi, "The implementation and evaluation of a disk scheduling policy for a WWW server based on its contents," *IPSJ Trans.*, vol.42, no.6, pp.1472–1482, 2001. (in Japanese)

[2] S. Suranauwarat and H. Taniguchi, "Operating systems support for the evolution of software: an evaluation using WWW server software," In *Proc. of the 2000 International Symposium on Principles of Software Evolution*, pp.289–298, 2000.