JOURNAL OF VISUAL COMMUNICATION AND IMAGE REPRESENTATION
Vol. 9, No. 3, September, pp. 183–193, 1998
ARTICLE NO. VC980395

Design and Performance Evaluation of a Multimedia Web Server†

Y. B. Lee*

Department of Computer Science, The Hong Kong University of Science and Technology, Hong Kong

and

P. C. Wong

Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong

Received November 17, 1997; accepted August 8, 1998

Today, vast amounts of text, images, graphics, animation, and even Java applets are being hosted and delivered by the WWW. With rapid advances in audio and video technologies, more and more contents will be encoded and delivered by means of audio/video in addition to texts and images. However, conventional Web servers are designed for data services and hence have no provisions for delivering continuous media such as audio and video. While one can still provide continuous-media services by a separate server in addition to the Web server, extra hardware cost and management complexity are incurred. Moreover, unused capacity in the Web server cannot be utilized even if the continuous-media server is overloaded, and vice versa. This paper presents a design and implementation of a server which integrates continuous-media services with traditional Web services. To resolve disk and network contentions, a simple yet effective fixed-priority scheduler is employed. Experimental results show that the proposed scheduler performs well with non-real-time hardware and operating system platforms. © 1998 Academic Press

1. INTRODUCTION

The tremendous success of the World Wide Web (WWW) has transformed the way information is retrieved and delivered in the Internet. The WWW is based on a client–server model where texts, hypertext (HTML), images, and graphics are stored at a Web server for delivery to Web browsers connected by a network. Today's web server software runs on a wide spectrum of hardware and operating systems, ranging from embedded systems to massively parallel supercomputers.

With rapid advances in audio and video technologies, more and more content will be encoded and delivered by means of audio/video in addition to texts and images. However, conventional Web servers are designed for data services and hence have no provisions for delivering continuous media (CM) such as audio and video. While a few commercial products allow streaming video directly from standard Web servers using the standard hypertext transfer protocol (HTTP), the video quality is usually limited and subject to the workload of the web server.

Another approach is to use a separate server to stream continuous media. Many commercial products are available in this category. Clearly this approach requires extra hardware and increases management complexity as multiple servers have to be maintained. Moreover, unused capacity in the Web server cannot be utilized even if the CM server is overloaded, and vice versa.

Unlike existing systems, this paper presents a design and implementation for a multimedia server where CM services are integrated with traditional Web services. In other words, the multimedia server software runs on a single server hardware and delivers both Web and CM services. We assume an Intranet environment such that the underlying network has sufficient bandwidth to carry the CM as well as Web traffic. By integrating the two services into a single piece of software, we can design an efficient buffer management scheme to enable buffer sharing. Moreover, we can use scheduling to resolve disk and network contentions so that delay-sensitive CM service can be protected from heavy Web traffic. Finally, we study how client congestion can be avoided by shaping traffic at the source.

The rest of the paper is organized as follows: Section 2 discusses some related works and compares them with our study; Section 3 presents the general architecture of the multimedia server; Section 4 describes how buffers are managed; Section 5 presents an I/O scheduling algorithm

* To whom correspondence should be addressed.
† This article is part of a special section devoted to the Image Technology for World-Wide-Web Applications.

for use at the disk and network; Section 6 studies traffic shaping at the source to reduce network and client congestion; Section 7 presents experimental results obtained from our implementation to evaluate the performance of the system designs; Section 8 summarizes the paper.

2. RELATED WORKS

There are many issues in delivering CM streams in the WWW platform. First, the client must be capable of receiving CM streams and presenting them using multimedia codecs in a synchronized manner. This can be done either by external helper applications, by plug-ins, or by having the functionality integrated within the browser [1]. Second, the TCP transport protocol employed in today's WWW does not perform well in delivering delay-sensitive, CM streams [1, 2]. Therefore much research has been conducted on designing new CM protocols. Some examples include the real time protocol (RTP) [3], video datagram protocol (VDP) [1], stream transfer protocol (STP) [2], and real time streaming protocol (RTSP) [4].

Third, the stringent timing requirements in CM retrieval pose new design challenges not found in traditional Web servers. While there are many studies on designing CM servers [2, 5–8], they do not include Web service. Conversely, we study an integrated server in this paper where CM service is integrated with the traditional Web service.

Another study by Z. Chen *et al.* [1] also designed and implemented an integrated server for delivering both types of services. Their extended server has separate handlers for HTTP and CM services. However, their focus is on the adaptation of CM traffic to avoid congestion in the Internet. Their VDP protocol employs feedback controls from their Vosaic client to adjust the frame rate transmitted at the server.

While bandwidth is a major problem in delivering continuous media over the Internet, bandwidth is less of an issue in Intranet environment, where high-speed networks such as switched Ethernet, FastEthernet, and ATM are readily available at low cost. Therefore the bottleneck shifts from the network to the server when one extends the web paradigm to corporate networks.

In this paper, we propose a simple yet effective scheduling algorithm for use in the disk and network subsystem to guarantee the quality of CM service despite heavy web traffic serving at the same server in an Intranet environment. Unlike previous studies [9–13], the proposed algorithm does not need real-time hardware and operating system support, and hence can be implemented and deployed in today's server platforms.

3. SYSTEM ARCHITECTURE

The general design of the system is shown in Fig. 1. The service manager is responsible for controlling individual services, including the stream service and the Web service. A standard interface is defined for the interaction between the service manager and the individual services so that new service can later be added without the need to change other system components. When a request arrives at the server, it is first processed by the appropriate service. Then one or more I/O requests will be generated by the service and submitted to the integrated disk scheduler (IDS). The IDS controls the queueing of the I/O requests and dispatches I/O requests to the disk manager for actual data retrieval. The scheduling algorithm employed will be described in Section 5.

After an I/O request completes service at the disk manager, it will be passed back to IDS for submission to the integrated transmission scheduler (ITS). Using the same

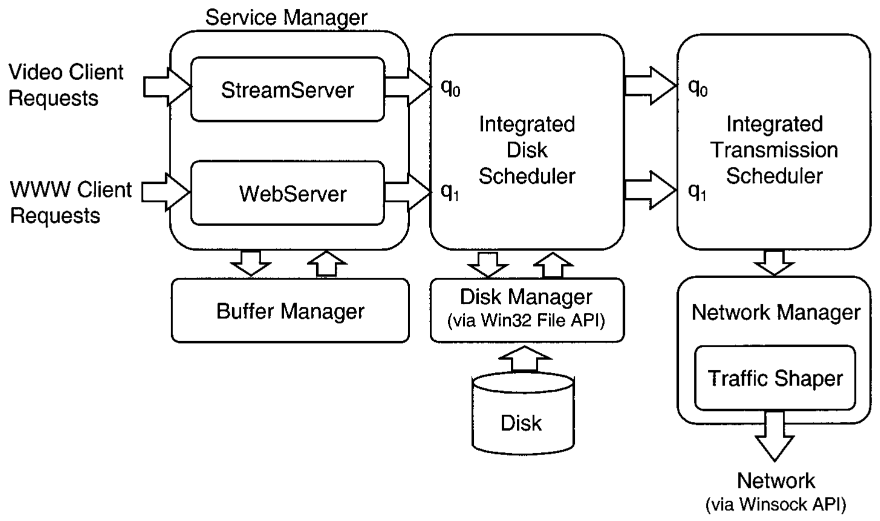

FIG. 1. System architecture of the multimedia Web server.

scheduling algorithm as in the IDS, the ITS controls the queueing of I/O requests for dispatch to the network manager. A traffic shaper (see Section 6) within the network manager then transmits data at a controlled data rate to avoid network and client congestion.

The stream service uses a client pull service model to control the flow of CM data from the server to a client. Specifically, a client periodically sends requests to the server to retrieve fixed-size data blocks for playback. CM data are delivered by a stream transfer protocol (STP) over UDP as described in [2]. To absorb variations in server response time and maintain audio/video playback continuity, the client prefetch a number of data blocks before starting playback. To avoid overloading the server, the Stream Service monitors the number of concurrent CM sessions and rejects new sessions if the maximum limit is reached. Interested readers are referred to [2] for details on system dimensioning.

The Web service implements the standard request–response HTTP protocol over TCP. Like commercial web servers, the Web service can handle multiple HTTP connections simultaneously. The maximum allowable number of concurrent HTTP connections can be configured during system initialization.

4. BUFFER MANAGEMENT

To reduce overhead in memory copies, we developed a custom buffer manager to manage a preallocated fixed pool of buffers. We use fixed-size buffers to simplify processing and reduce allocation complexity. Moreover, the same buffer pool is shared by all services in the system, i.e., the stream service and the Web service both allocate buffer from the same buffer manager. This buffer-sharing approach reduces the total number of buffers needed at the server. To further improve performance, only buffer pointers are passed from one component to the next to avoid memory copies. Experimental results (see Section 7.1) show that our buffer manager outperforms the buffer manager provided by the operating system and the compiler.

5. I/O SCHEDULING

Multimedia scheduling has been studied extensively in the literature, such as in [9–13]. While the proposed algorithms can guarantee real-time performance, they in turn require real-time support from the underlying operating system and computer hardware. In practice, almost all web servers in use today run on general-purpose computers and operating systems where real-time support is not available. Therefore we design a new scheduler for scheduling CM and data traffic without the need for real-time support. Experimental results (Section 7.2) show that the scheduler can effectively protect CM streams from heavy data traffics.

5.1. *Scheduling Algorithm*

There are two types of resource contention problems in a multimedia server: (a) Requests queueing—time-critical requests such video can be delayed by non-time-critical requests such as data requests which arrived earlier; (b) Requests blocking—time-critical requests can be blocked by a single large data request inside service.

To solve the first problem, we use multiple static-priority queues with concurrency control to assign a higher priority to time-critical requests. There are N_{Q} queues in the scheduler, denoted as $q_0, q_1, \ldots, q_{N_{\mathrm{Q}}-1}$. Let queue i be assigned a unique fixed priority π_i, with a lower value representing a higher priority. The requests are dispatched from the queues according to the queue's priority. Each queue has a maximum concurrency limit M_i, which governs how many requests of that priority can be serviced simultaneously by the server. Finally, there is a maximum concurrency limit M_{D} to limit the total number of requests of any priorities that can be served concurrently.[1]

To solve the second problem, we set a limit on the maximum size of each request. Requests for large data blocks are divided into smaller subrequests of maximum size Q and they are submitted to the scheduler one by one. This allows the scheduler to schedule requests at a granularity of Q bytes, even if the original request is larger than Q. Since the request breakup is done within the server, the processing is transparent to the client.

Figure 2 lists the integrated scheduling algorithm using C-like pseudocode. There are two points worth noting. First, the concurrency of the device and the corresponding queue will be decremented when a request completes service. This is not shown in the pseudocode because I/O completion events occur asynchronously. Second, the algorithm resets i to zero after submitting a request to the device to guarantee that lower-priority requests in queues are not served unless all higher priority queues either have no requests in queue or have already reached their concurrency limits.

5.2. *Integrated Disk Scheduler*

The Integrated Disk Scheduler is an implementation of the integrated scheduler for scheduling requests in the disk subsystem. The number of queues and their priorities are assigned at system start-up by the system manager. For example, the system manager may prefer that 60% of the disk throughput be reserved for video and audio, and 40% for data traffic. In our implementation, we have two prior-

[1] Note that M_i and M_{D} are not limits for the length of the queues. They limit the number of outstanding requests only.

```
Algorithm:
While (not terminate)
{
 for ($i$=0; $i$<$N_Q$; $i$++)
 {
  if (concurrency of device = $M_D$)
  {
   wait for one request to complete service.
   Reset $i$=0.
  }
  if ($q_i$ is not empty and
      concurrency of $q_i$<$M_i$)
  {
   Remove one request from $q_i$,
    submit request to device,
    increment concurrency of device and $q_i$
   Reset $i$=0.
  }
 }
}
```

FIG. 2. The integrated scheduling algorithm.

ity queues: q_0 (high priority) is assigned for stream type traffic, and q_1 (low priority) is assigned for web traffic. As our disk is scheduled using C-SCAN[2] [10] within the operating system, we have chosen a maximum concurrency of $M_D = 10$ to minimize disk seek overhead. To allocate 60% of disk throughput to stream-type traffic, we assign $M_0 = 6$ and $M_1 = 4$ for q_0 and q_1, respectively.

5.3. *Integrated Transmission Scheduler*

The Integrated Transmission Scheduler (ITS) schedules and dispatches transmission requests to the traffic shaper for transmission. Again we use two priority queues: q_0 (highest priority) is assigned for stream-type traffic, and q_1 is assigned for data traffic. In the traffic shaper, we have 10 send queues, so we set $M_D = 10$.

6. TRAFFIC SHAPING

As described in the previous section, our server retrieves data in fixed-size blocks to facilitate I/O scheduling. Obviously using larger block sizes can reduce processing overhead and increase I/O efficiency. This is especially significant in the disk subsystem where seeking and rotational latency overhead is incurred in every I/O transaction.

While using large blocks can improve efficiency, the client may become congested if a server transmits an entire block of packets in a very short time. This is especially true if the server is faster than the client, or the server is using a higher-speed link. The instantaneous rate of transmitting one entire block can be very high. This problem is significant in practice because the clients are usually medium-grade, general-purpose computers. They may be busy in other tasks such as video decoding or display updating, so they may not be able to keep up with the incoming packets. Consequently some packets may be dropped at the client, causing unnecessary retransmissions [2]. Therefore we need to control the burstiness of server transmissions.

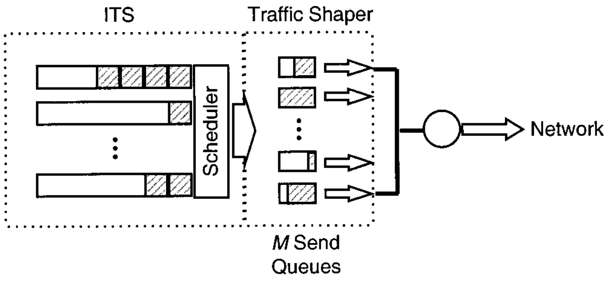

FIG. 3. The traffic shaper.

The traffic shaper is depicted in Fig. 3. There are M send queues in the traffic shaper. Each send queue serves one (web or CM) data block at a time. Assume that m out of the M send queues are active, that is, have data blocks for transmission. In each round of transmission, the server will transmit g packets, each packet of Y bytes from each of the active queues in a round-robin manner. Hence gY bounds the burst size, and the server will transmit gm packets in each round of service. The instantaneous rate r of packet transmissions to a client will be S_S/m, where S_S is the server throughput in bytes per second. As some of the send queues may be empty, we define a minimum round size $M_{\min}$. If $m < M_{\min}$, the server will serve the m active send queues first and then suspend service for a time of $(M_{\min} - m)gY/S_S$ until the end of a round, as if it is serving $M_{\min}$ queues in this round. Therefore the maximum instantaneous transmission rate to any client will be $r_{\max} = S_S/M_{\min}$. On the other hand, the minimum transmission rate is $r_{\min} = S_S/M$.

This traffic shaper allows us to control the minimum rate, the maximum rate, and the size of each burst of packets transmitted. Experimental results (Section 7.3) show that the traffic shaper can significantly reduce packet loss at the client stations.

7. PERFORMANCE EVALUATION

To evaluate the performance of the proposed system designs, we developed a multimedia web server using C++

[2] C-SCAN (also called elevator seeking) is a disk-scheduling algorithm where the disk arm moves back and forth between the innermost track and the outermost track, reading data along the way to minimize seeking overhead.

TABLE 1
System Configurations for the Multimedia Web Server

Item	Configurations
Server computer	Pentium-90, 256KB SRAM, 32MB DRAM, SCSI-II 2GB HD
Client computer	i486 and P5 PCs
Network adapter	Intel EtherExpress Pro/100 (PCI), 3Com Fast-Etherlink 3C590 (PCI), Intel EtherExpress 10 (ISA)
FastEthernet switch	BayNetworks 28115 (16 100Mbps Ports)
Hardware MPEG video decoder	Sigma Designs RealMagic (MPEG-1)
Server OS	Microsoft Windows NT 3.51
Client OS	Microsoft Windows for Workgroup 3.11, Windows 95, Windows NT 3.51, 4.0.
WWW browser	Netscape Navigator, Microsoft Internet Explorer, and WebStone (benchmarking)

on Microsoft's Windows NT operating system. The server runs as a multithreaded user-mode application. We use the Windows Socket API for network I/O and the Win32 API for file I/O. No low-level access to the network or disk drivers is employed. The server hardware is a Pentium 90-Mhz PC with 32 MB RAM and SCSI-II hard disks. The detailed system configuration is listed in Table 1. We use WebStone 2.0 [14] to generate HTTP requests according to the standard retrieval distribution shown in Fig. 4. For the stream service, requests are generated from 10 PCs using the video client as described in [2] with 540-KB buffers. Each client PC establishes an active video connection for viewing a 1.2-Mbps MPEG 1 video, which is a 30 fps, VCR quality video. We determine the server capacity by increasing the number of concurrent video streams until video starvation occurs, i.e., failure to play back video continuously at the prescribed frame rate.

7.1. *Buffer Manager*

To evaluate the performance of our buffer manager, we benchmarked the time taken to allocate and deallocate memory objects of various sizes using our custom buffer manager as well as the compiler's built-in memory manager (*new* and *delete* under C++). Note that the compiler's built-in memory manager calls the operating system's memory manager for memory allocation and then suballocates memory to applications. The results are plotted in Fig. 5, which clearly shows that our custom buffer manager outperforms the compiler's built-in memory manager by 190% to 788% for allocations, and by 158% to 315% for deallocations. For buffer size 64 KB (used extensively in our prototype), our custom buffer manager is 428% faster for allocation and 192% faster for deallocation.

There are two reasons for the performance improvement. First, our buffer manager allocates a pool of buffers at startup and hence does not need to call any operating system functions for future memory allocations. Second, our custom buffer manager handles fixed-size memory objects and hence eliminates complex buffer allocation algorithms as used in the compiler's general-purpose memory manager.

While the actual performance will vary from one platform to another, general performance gains can be expected as our optimizations are generic and free from platform-specific tweakings. For example, running the same

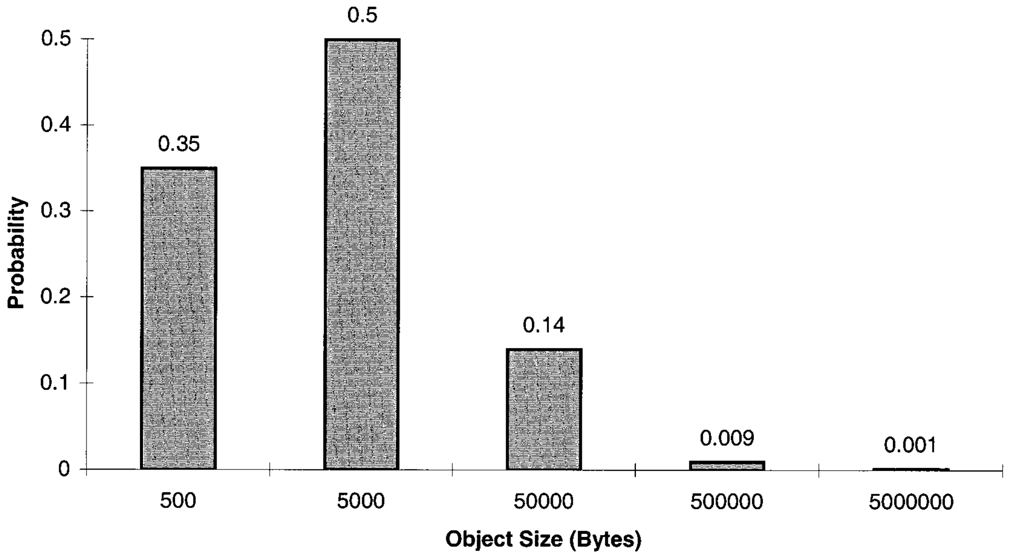

FIG. 4. The WebStone standard retrieval distribution.

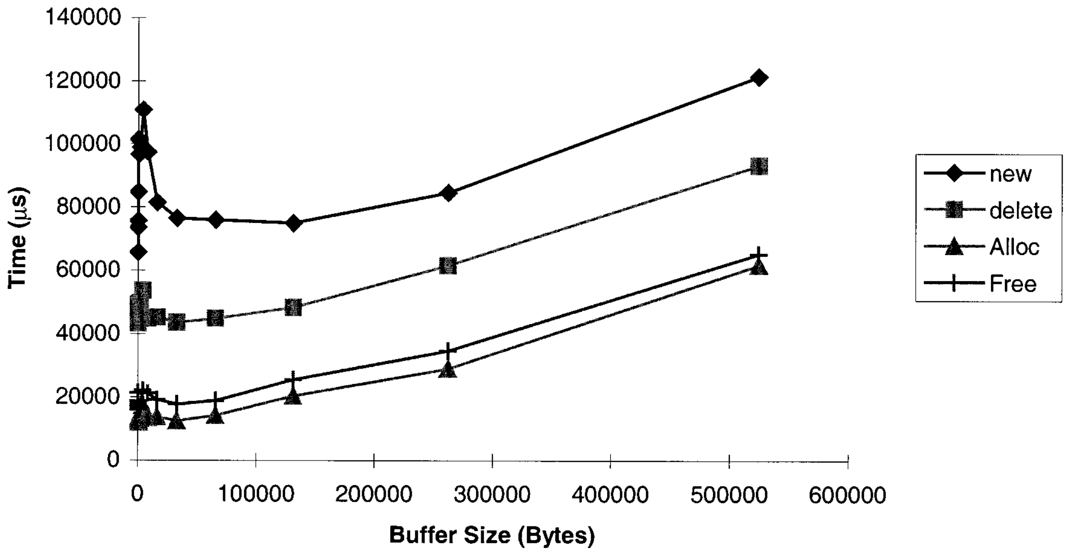

FIG. 5. Buffer manager performance comparison (time shown for 10,000 iterations).

benchmark under SunOS 4.1.3 on a SPARC10 workstation, our custom buffer manager is 348% and 314% faster for allocation and deallocation of 64-KB buffers, respectively.

7.2. I/O Schedulers

In this section we analyze the performance of the Integrated Disk Scheduler and the Integrated Transmission Scheduler in scheduling continuous-media and data services. We run benchmarks to measure the throughput of both services by varying the number of video sessions while concurrently serving 20 Web clients simulated using WebStone 2.0 with the standard file set and access distribution. The results are summarized in Fig. 6.

We observe that the Web service throughput drops progressively as more video sessions are established. On the other hand, the stream service throughput increase proportionally with the number of video sessions. This demonstrates the effectiveness of the Integrated Disk Scheduler and the Integrated Transmission Scheduler in giving priority to STP service.

In order to study the effectiveness of each scheduler, we ran the same test to collect the mean system time (service plus queueing) for both services using the following configurations:

• *None*—No scheduling at disk and network
• *NetOnly*—The network scheduler is enabled while the disk scheduler is disabled
• *DiskOnly*—The disk scheduler is enabled while the network scheduler is disabled
• *Both*—Both the disk and network schedulers are enabled.

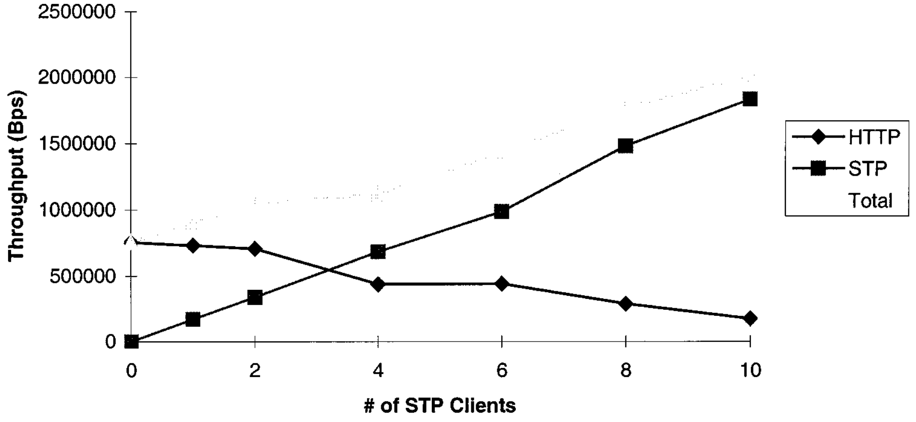

FIG. 6. Stream versus Web throughput for varying stream loading.

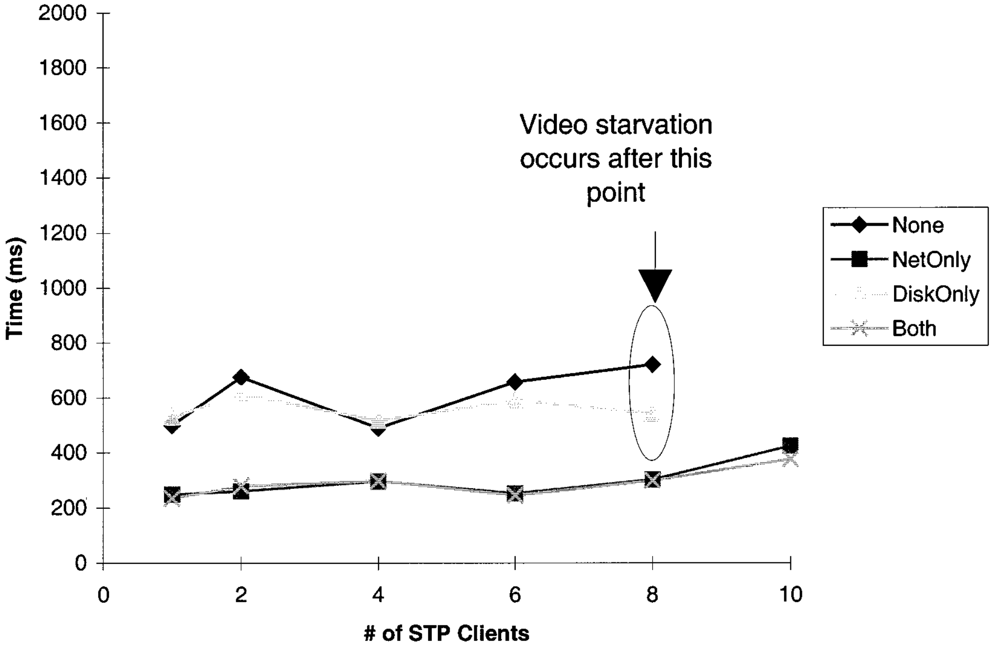

FIG. 7. Mean system time for stream service with 20 Web clients and varying stream loading.

Figures 7 and 8 plot the mean system times for the video and Web services, respectively. First consider the case of both schedulers enabled. We see that while the Web service's mean system time increases with increasing video loading, the video service's system time remains below 400 ms regardless of the server loading. This again suggests that the I/O schedulers indeed protect the video service from Web loading at the expense of increased system time in Web service.

Second, if we compare the four cases, it may appear that the network scheduler alone is sufficient for protecting the video service. However, further investigation reveals that

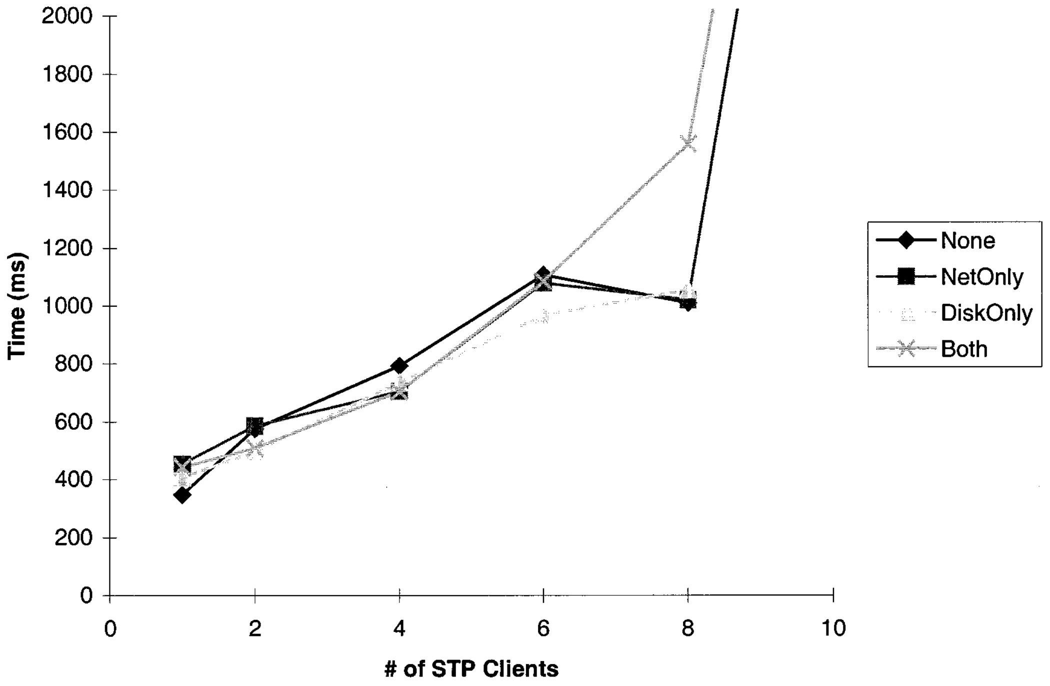

FIG. 8. Mean system time for Web service with 20 Web clients and varying stream loading.

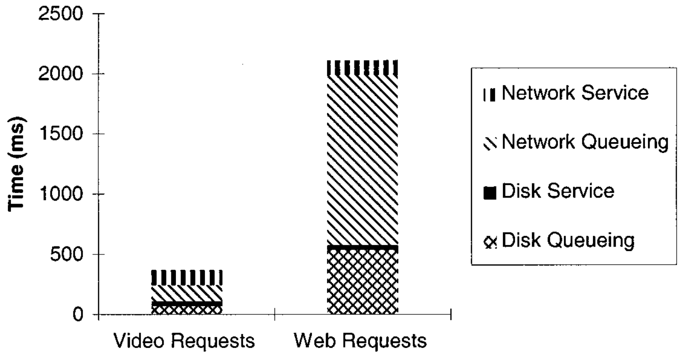

FIG. 9. System time compositions for stream service and Web service at peak loading.

this is true only if the network subsystem is the bottleneck. To see why, we consider the relative amount of time spent in each stage of the server at peak loading. The system time composition is charted in Fig. 9. For both services, the time spent in the network subsystem (network queueing time and service time) constitutes 71% and 73% of overall system time for video and Web services, respectively. This explains why the network scheduler is effective while the disk scheduler is ineffective.

To further confirm our conjecture, we conduct another test where external loading is applied to the disk subsystem. This load is generated by an independent process periodically reading data blocks from the same disk where video and WWW data are stored. In our test the process is set to read data at 5 Mb/s.

The mean system times are plotted in Figs. 10 and 11 for video and Web services, respectively. Note that the system can only support four concurrent video streams without the disk scheduler. Using the scheduler extends the capacity to six concurrent video streams. Studying the system time compositions in Fig. 12 reveals that the disk becomes the bottleneck in this test due to the external disk loading.

Therefore both the disk scheduler and network scheduler are necessary to protect the CM service in a general system. On the other hand, to ensure that Web sessions are not totally denied service, it is crucial to limit the maximum number of CM sessions on a server by some admission control algorithms. This ensures that the system will have a sufficiently large spare capacity to provide a reasonably good web service.

7.3. *Traffic Shaper*

As discussed in Section 6, traffic shaping can reduce the amount of packet dropping at the network and client due to congestion. To evaluate the effectiveness of the traffic shaper, we measured the packet loss probability at the client for various values of granularity g at the traffic shaper and summarized the results in Table 2. The results confirm that the traffic shaper can effectively reduce packet dropping at the client. By using an interleaving granularity of one, the packet lost probability is less than 0.5%, compared with the 10% loss for $g = 4$. We could not obtain reliable results for larger values of g as the loss is too severe for the video player to correctly continue playback.

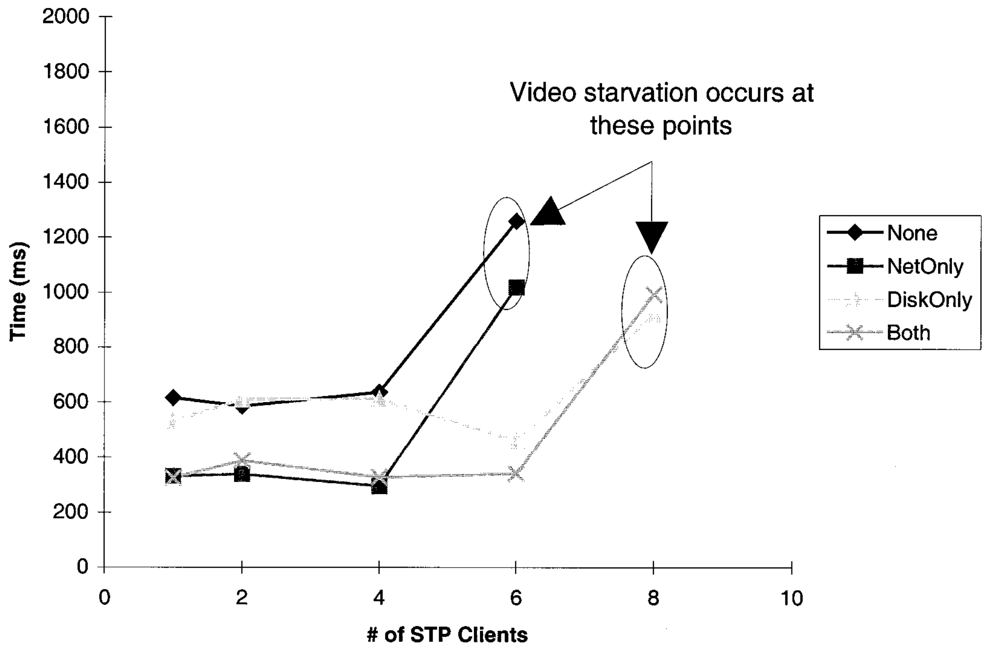

FIG. 10. Mean system time for stream service with 20 Web clients and varying stream loading with 5 Mb/s external disk load.

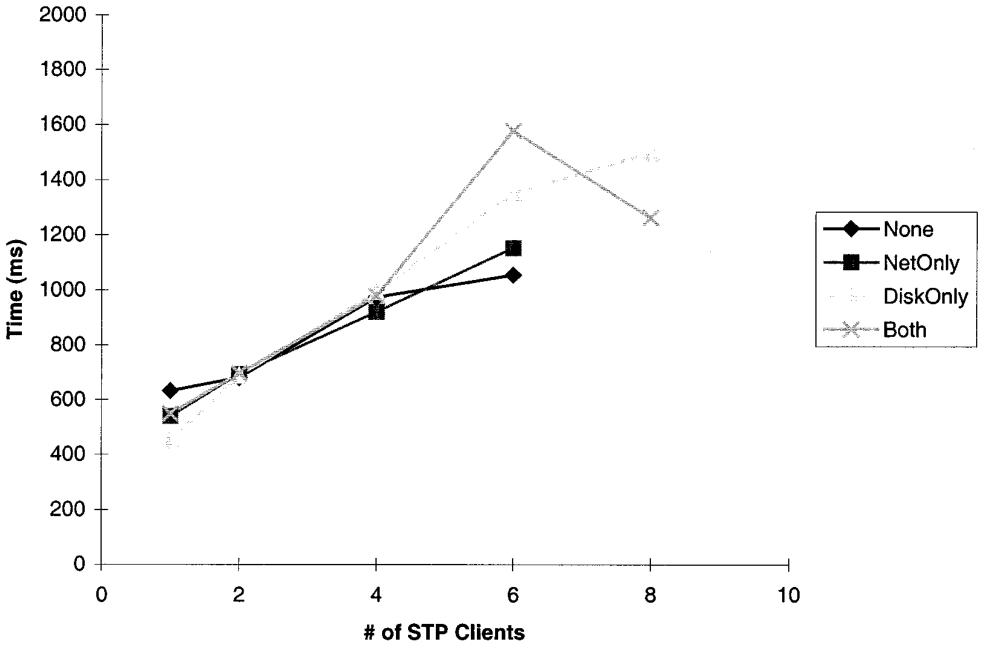

FIG. 11. Mean system time for Web service with 20 Web clients and varying stream loading with 5 Mb/s external disk load.

We also collected the packet interarrival times at the client during the same tests. The distributions are shown in Fig. 13 for $g = 1$, Fig. 14 for $g = 2$, and Fig. 15 for $g = 4$. Note that the average interarrival time increases (hence smoother traffic) for smaller values of g. This explains why the packet loss probability can be reduced by the traffic shaper.

8. CONCLUSION

In this paper, we presented the design and implementation of a multimedia server where continuous-media service is integrated with Web service into a single system. First, by integrating the two services, we designed a scheduler to resolve resource contentions in the disk and network subsystem. Our experimental results showed that the scheduler can effectively protect the CM service despite heavy Web traffic. Second, we designed a custom buffer manager for sharing buffers among the two services. This buffer manager not only reduced the total buffer requirement but also outperformed the compiler's built-in buffer manager in terms of allocation and deallocation time. Third, we employed a traffic shaper at the network to control the burstiness of outgoing traffic. Experimental results showed that shaping network traffic at the source can effectively prevent congestion at the client. The system design presented in this paper does not require real-time support from the operating system and server hardware, and hence can readily be incorporated into today's web server software.

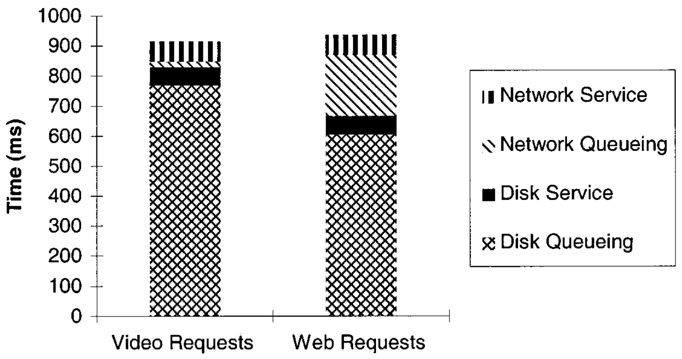

FIG. 12. System time compositions for stream service and Web service at peak loading with 5 Mb/s external disk load.

TABLE 2
Packet Loss Probabilities versus Traffic Shaping Granularity

Traffic shaping granularity (packets)	Packet loss (percent)
4	10%
2	1%
1	<0.5%

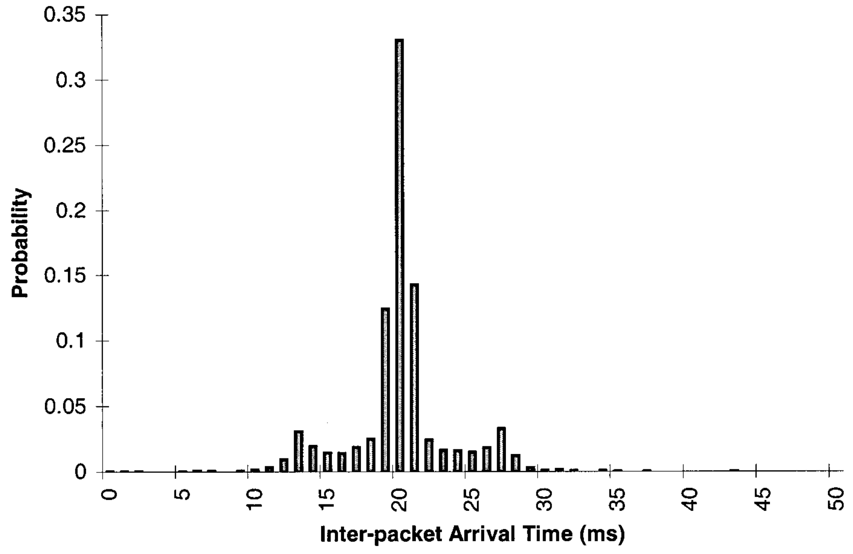

FIG. 13. Packet interarrival time distribution at the client for traffic shaping granularity of $g = 1$.

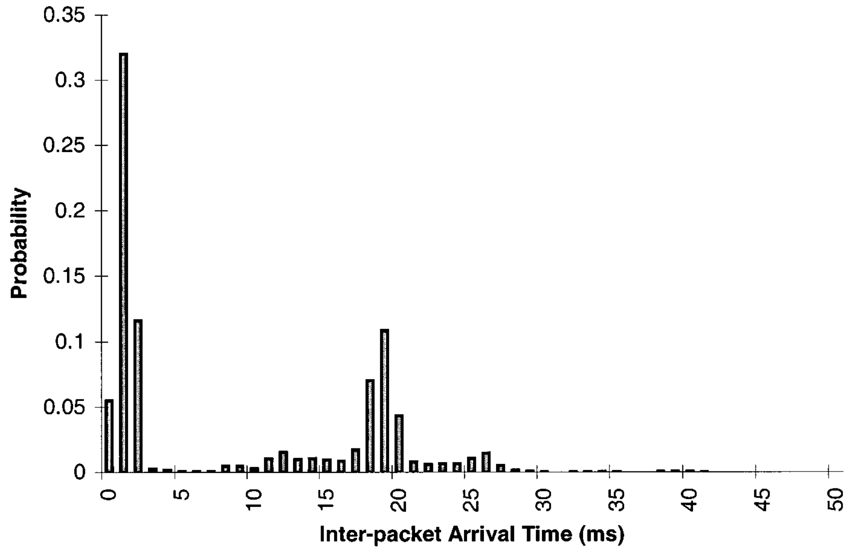

FIG. 14. Packet interarrival time distribution at the client for traffic shaping granularity $g = 2$.

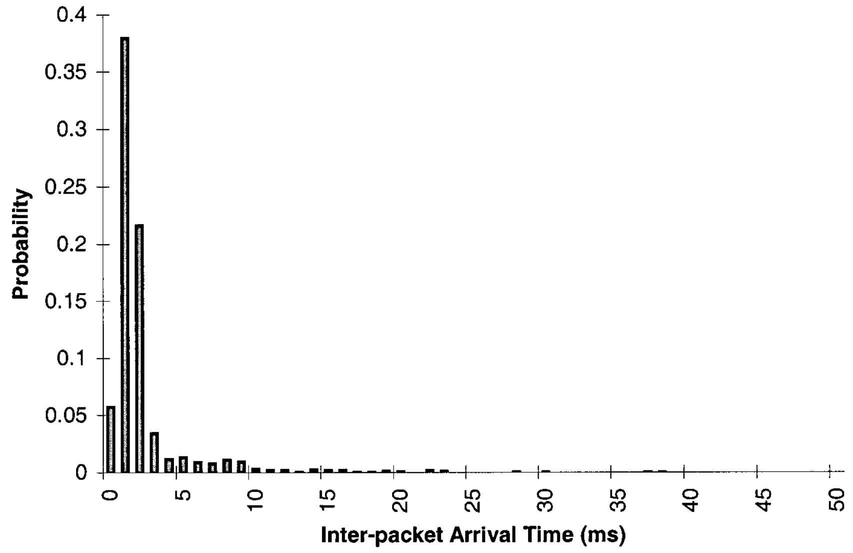

FIG. 15. Packet interarrival time distribution at the client for traffic shaping granularity of $g = 4$.

REFERENCES

1. Z. Chen, S. M. Tan, R. H. Campbell, and Y. C. Li, Real time video and audio in the World Wide Web, *World Wide Web J.* **1**(1), 1996.
2. Y. B. Lee and P. C. Wong, A server array approach for video-on-demand service on local area networks, in *IEEE INFOCOM 1996, San Francisco*.
3. RTP—A transport protocol for real-time applications, in *RFC 1889, 1996*.
4. *Real Time Streaming Protocol*, Internet Draft—draft-ietf-mmusic-rtsp-05.txt, October 1997. [Available: http://brinkley.prognet.com/rtsp/protocol.txt]
5. F. A. Tobagi and J. Pang, StarWorks—A video applications server, in *IEEE COMPCON Spring 1993*, pp. 4–11.
6. D. Jadav and A. Choudhary, Designing and implementing high-performance media-on-demand servers, *IEEE Parallel Distrib. Technol. Systems Appl.* **3**(2), 1995, 29–39.
7. B. Ozden, R. Rastogi, and A. Silberschatz, On the design of a low cost video-on-demand storage system, *ACM Multimedia Systems* **4,** 1996, 40–54.
8. A. Klemets, The design and implementation of a media on demand system for WWW, in *Proc. 1st International Conference on the World-Wide Web, Geneva, 1994*. [Available http://www.it.kth.se/~klemets/www.html]
9. D. J. Gemmell, H. M. Vin, D. D. Kandlur, P. V. Rangan, and L. A. Rowe, Multimedia storage servers: A tutorial, *IEEE Comput.* **28**(5), 1995, 40–49.
10. A. L. N. Reddy and J. C. Wyllie, I/O issues in a multimedia system, *IEEE Comput.* **27**(3), 1994, 69–74.
11. D. R. Kenchammana-Hosekote and J. Srivastava, Scheduling continuous media in a video-on-demand server, in *Proc. International Conference on Multimedia Computing and Systems, Boston, 1994*.
12. D. D. E. Long and M. N. Thakur, Scheduling real-time disk transfer for continuous media applications, in *Proc. 12th IEEE Symposium on Mass Storage Systems, 1993*, pp. 227–232.
13. S. J. Daigle, Disk scheduling for multimedia data systems, in *Proc. 1994 SPIE Conference on High-Speed Networking and Multimedia Computing*, 1994.
14. WebStone 2.0, SGI. Available http://www.sgi.com/Products/WebFORCE/WebStone.

JACK YIU-BUN LEE received his B.Eng. and Ph.D. from the Department of Information Engineering of the Chinese University of Hong Kong in 1993 and 1997, respectively. After spending one year as a visiting assistant professor at the same department, he joined the Department of Computer Science at the Hong Kong University of Science and Technology as an assistant professor in 1998. His research interests include distributed multimedia-on-demand systems, fault-tolerant systems, and Internet computing. Lee is a member of IEEE and ACM. He can be reached at jacklee@computer.org.

P. C. WONG is currently associate professor at the Department of Information Engineering at the Chinese University of Hong Kong. Before that, he was a lecturer in the Department of Computing Studies of Hong Kong Polytechnic for five years, a visiting scientist at the IBM T.J. Watson Center, New York, in 1992, a visiting research professor in the Department of Communication Engineering of National Chiao Tung University in 1993, and a visiting research fellow in the Cooperative Research Centre of Telecommunication and Broadband Networking at Curtin University of Technology, Western Australia, in 1995. Wong is a senior member of IEEE and can be reached at pcwong@ie.cuhk.edu.hk.