Performance Analysis of a Pull-Based Parallel Video Server

Jack Y.B. Lee, *Member*, *IEEE*, and P.C. Wong, *Senior Member*, *IEEE*

Abstract—In conventional video-on-demand systems, video data are stored in a video server for delivery to multiple receivers over a communications network. The video server's hardware limits the maximum storage capacity as well as the maximum number of video sessions that can concurrently be delivered. Clearly, these limits will eventually be exceeded by the growing need for better video quality and larger user population. This paper studies a parallel video server architecture that exploits server parallelism to achieve incremental scalability. First, unlike data partition and replication, the architecture employs data striping at the server level to achieve fine-grain load balancing across multiple servers. Second, a client-pull service model is employed to eliminate the need for interserver synchronization. Third, an admission-scheduling algorithm is proposed to further control the instantaneous load at each server so that linear scalability can be achieved. This paper analyzes the performance of the architecture by deriving bounds for server service delay, client buffer requirement, prefetch delay, and scheduling delay. These performance metrics and design tradeoffs are further evaluated using numerical examples. Our results show that the proposed parallel video server architecture can be linearly scaled up to more concurrent users simply by adding more servers and redistributing the video data among the servers.

Index Terms—Parallel video server, striping, performance analysis, admission scheduling, scalable, client pull.

1 Introduction

In a video-on-demand (VoD) system, digitized and compressed video streams are stored in a video server for delivery to receivers over a communications network. While VoD systems are already commercially available today, they are usually designed around the single-server architecture where a single machine serves as the video server. The video server can range from a standard PC for small-scale systems [1], [2] to massively-parallel supercomputers with thousands of processors for large-scale systems [3], [4].

Due to the lack of economy of scale, the price/performance ratio tends to increase rapidly for high-end servers. This makes the single-server approach expensive for large-scale applications. Moreover, the capacity of a single server is still ultimately limited and, hence, replication is often needed to build systems with sufficient capacity [5], [6].

This motivates us to investigate the design of VoD systems using parallel-server architectures. Server-level parallelism enables one to break through the limit of a single server by aggregating the capacity and bandwidth of multiple servers. Moreover, this parallel-server architecture allows us to incrementally scale up the system capacity by adding (rather than replacing) servers to the system and then redistributing (rather than replicating) the video data among the servers.

• *The authors are with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T. Hong Kong. E-mail: jacklee@computer.org., pcwong@ie.cuhk.edu.hk.*

Manuscript received 10 Nov. 1997; revised 16 Oct. 1998; accepted 11 Mar. 1999.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 105913.

The contributions of this paper are:

a. We propose a system design based on parallel-server architecture to build incrementally scalable VoD systems.
b. We study the use of a client-pull service model which completely avoided the need for interserver synchronization.
c. We study the instantaneous load imbalance problem arising from the parallel architecture and propose a staggering-based admission scheduler that enables the system to be linearly scalable.
d. We derive a performance model for the parallel video server to obtain various performance metrics such as server service delay, client buffer requirement, and admission scheduling delay.
e. We evaluate the performance of the proposed architecture using numerical results and confirm that the architecture is indeed linearly scalable.

The rest of the paper is organized as follows: Section 2 presents the system architecture, Section 3 addresses the load imbalance issue and presents the admission scheduler, Section 4 analyzes the server's worst-case service delay, Section 5 addresses the video playback continuity problem, Section 6 presents numerical results for various performance metrics, Section 7 discusses some related works and compares them with our study, and Section 8 concludes the paper and discusses some future works.

2 System Architecture

A parallel video server is composed of multiple independent servers connected to the client hosts by an interconnection network as shown in Fig. 1. Each server has a separate CPU, memory, disk storage, and network interface. Ideally, each server should be configured so that all

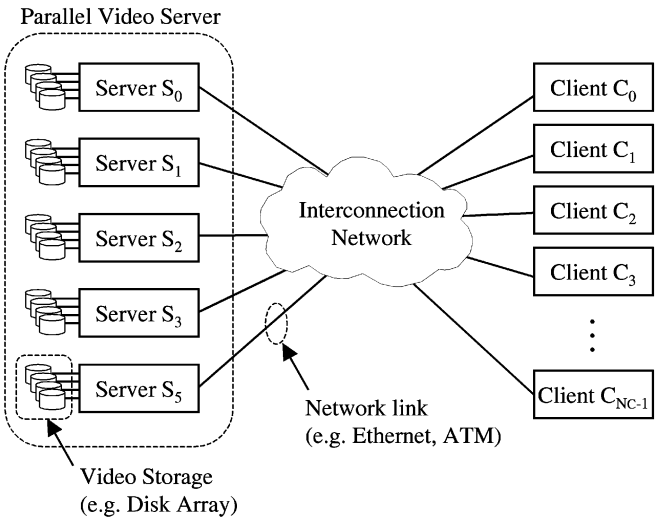

Fig. 1. Architecture of a parallel video server.

resources are fully utilized. For example, one would use a disk array with enough disks to fully utilize the disk-controller I/O capacity, install multiple network interfaces to fully utilize the CPU, or even use multiple CPUs to fully utilize the system bus if it is cost-effective. The server-level parallelism would provide the path for capacities beyond what can be achieved from a single server.

Unlike server replication or partition, a video title is not stored in one of the server nodes nor replicated over multiple server nodes. The video title is divided into fixed-size blocks and distributed over all server nodes in the system (Section 2.1). In this paper, we assume that the size of a video title is much larger than the size of a video block so that load imbalance due to uneven allocation between servers can be ignored. This assumption is generally applicable for movie applications where the size of a video title is in the range of hundreds of megabytes while the size of a video block is only in the range of tens to hundreds of kilobytes. More investigation will be needed for applications where extremely short video objects are striped in systems with a very large number of servers.

A client knowing the placement policy of the video blocks will send requests to each server to retrieve the required video blocks (Section 2.2). The servers upon receiving client requests will retrieve and transmit the requested data back to the client (Section 2.3). The client will resequence video packets from multiple servers for decode and playback. Note that this parallel-server architecture can be extended to include data redundancies so that server-level failure can be sustained (Section 2.4).

Let N_S be the number of servers and N_C be the number of clients in the system. The ultimate goal of the proposed parallel video server architecture is to achieve linear scalability, i.e., invariant client-server ratio when more servers are added to the system. Note that as storage requirement stays the same regardless of the scale of the system, the cost-per-client actually decreases when scaling up the system.

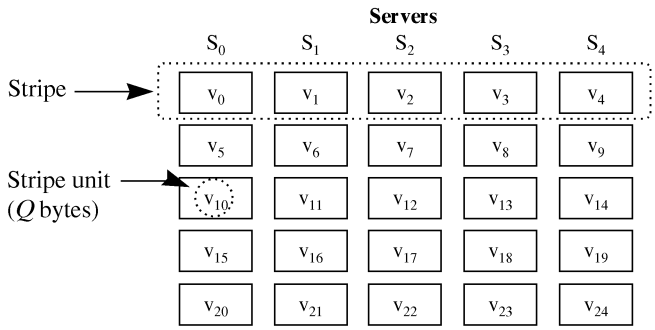

Fig. 2. Server striping for a five-servers parallel video server.

2.1 Server Striping

Fig. 2 shows the organization of stripe units among servers in the system. We divide storage space in each server into fixed-size stripe units of Q bytes each. Stripe units under the same column are stored at the same server. A row of stripe units is called a stripe and together all stripe units compose the entire storage. The stripe units are ordered in a round-robin manner in which each unit in a stripe belongs to a different server. In this way, the servers uniformly share client loads irrespective of the skewness of the video titles. Under this striping scheme, the placement policy can be completely described by three parameters: 1) the address of the server that stores the first video block, 2) the number and address of all servers, and 3) the size of a stripe unit. The client can obtain these parameters easily by consulting a database or other application-dependent methods.

2.2 Service Model

The server-push service model is common in single-server based video server designs. Under this model, the server schedules the periodic retrieval and transmission of video data once a video session is started. This model allows one to design schedulers to optimize disk and network utilization. However, extending this server-push service model for use in a parallel video server requires synchronizing transmissions from multiple servers destined to the same client, which is a nontrivial problem in its own right.

Unlike most studies, we consider the client-pull service model in this paper where a client periodically sends requests to the servers to retrieve blocks of video data. This approach does not need explicit interserver synchronization because each request is served at the server independently of all other requests. We will show that this approach can be linearly scalable even though the servers are completely asynchronous.

2.3 Server Request Processing

At each of the server node, there are one request queue, one shared queue, and M send queues organized as shown in Fig. 3. Incoming requests are first stored into the request queue. The disk retrieval process then serves requests in batches by allocating free buffers and reading video blocks from the disk into the buffer units. Requests within a batch may be served out of order to optimize disk efficiency. However, requests are grouped according to their arrival time so that no request is starved of service. After retrieval,

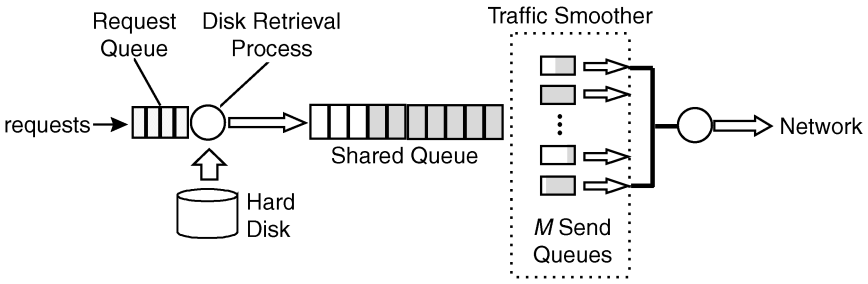

Fig. 3. Request processing at each individual server.

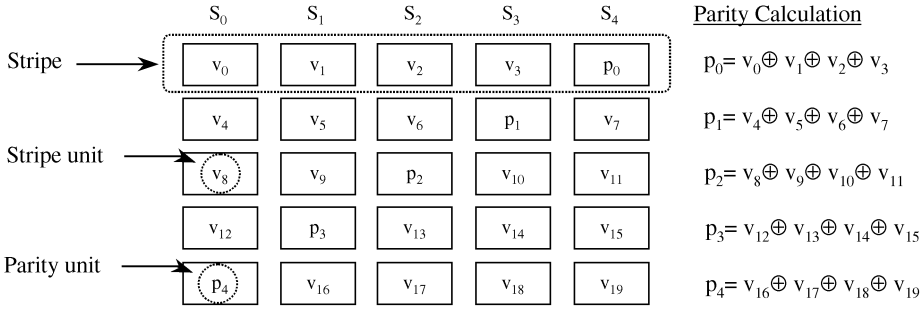

Fig. 4. Striping with redundancies for server-level fault tolerance.

these buffers are then pushed into the shared queue to wait for transmission at the traffic smoother.

The traffic smoother is used to control the transmission rate at the server to reduce outgoing traffic burstiness. Specifically, a video block inside a send queue is packetized into smaller packets of Y bytes each for transmission. The nonempty send queues are served in a round-robin manner with one packet transmitted from each send queue in a service round. The number of send queues M are chosen such that the transmission rate is high enough to sustain the video playback rate but not too high to avoid network congestion. For example, suppose the network throughput is 20 Mbps and the video bit-rate is 1.2 Mbps. Then setting $M = 10$ will result in a transmission rate of 2 Mbps, which is higher than the video bit-rate but still substantially lower than the maximum transmission rate that can cause network congestion (e.g., the client may be connected via a 4 Mbps link). Interested readers are referred to [7] for more details.

Unlike traditional round-robin scheduling scheme that needs one send queue per client, this algorithm needs only a fixed number of send queues irrespective of the number of servers and clients. This unique property prevents the server complexity from going up when scaling up the system for more servers and clients.

2.4 Fault Tolerance

Similar to disk arrays, the striping of video data over multiple servers also presents reliability problem. Specifically, the entire system will become inoperable if one or more servers in the system fail. Worse still, the larger the system (i.e., more servers), the less reliable it becomes. Fortunately, one can extend ideas from the disk-array context, i.e., Redundant Array of Inexpensive Disks (RAID) [8], to the server level to achieve server-level fault tolerance [9].

Specifically, we can introduce redundant data blocks among the servers as shown in Fig. 4. These redundant data blocks are computed from the other data blocks using simple XOR operations (or more sophisticated encoding for higher levels of redundancies). In case a server fails, the client can then recover the unavailable video blocks using the redundant blocks and the remaining data blocks as shown in Fig. 5. A preliminary study [9] has shown that this recovery could be done in real-time using software running under conventional desktop computers. With additional buffering at the client, video playback continuity can be maintained despite server failure and the user will not even know that a failure has occurred. Interested readers are referred to [9] for details on the redundant striping policy, the redundant transmission policy, and the failure-detection protocols, as well as experimental results.

3 Admission Scheduling

Server striping ensures that loads from each and every client will be evenly shared across all servers on the average. However, if we allow clients to start video sessions at arbitrary times, the system may encounter instantaneous load imbalance if many clients happen to start video sessions synchronously. In the worst case, all clients will send requests to the same server at the same time, possibly temporary overloading that server.

To prevent instantaneous load imbalance, we can explicitly schedule the start times for new video sessions to avoid synchrony. To achieve this, we introduce an admission scheduler to control when a client can start a new video session. This admission scheduler is a software process running on a computer host connected to the same network as the clients. We present the scheduling algorithm used in this admission scheduler in the next section.

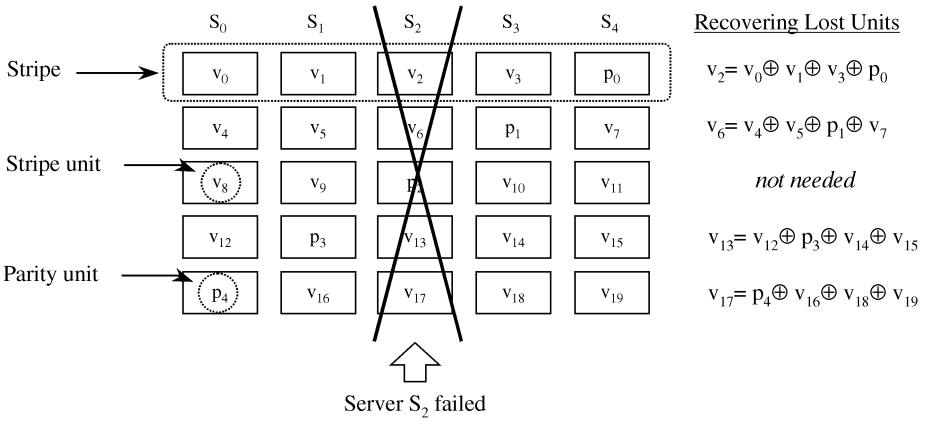

Fig. 5. Recovering unavailable data blocks by erasure correction.

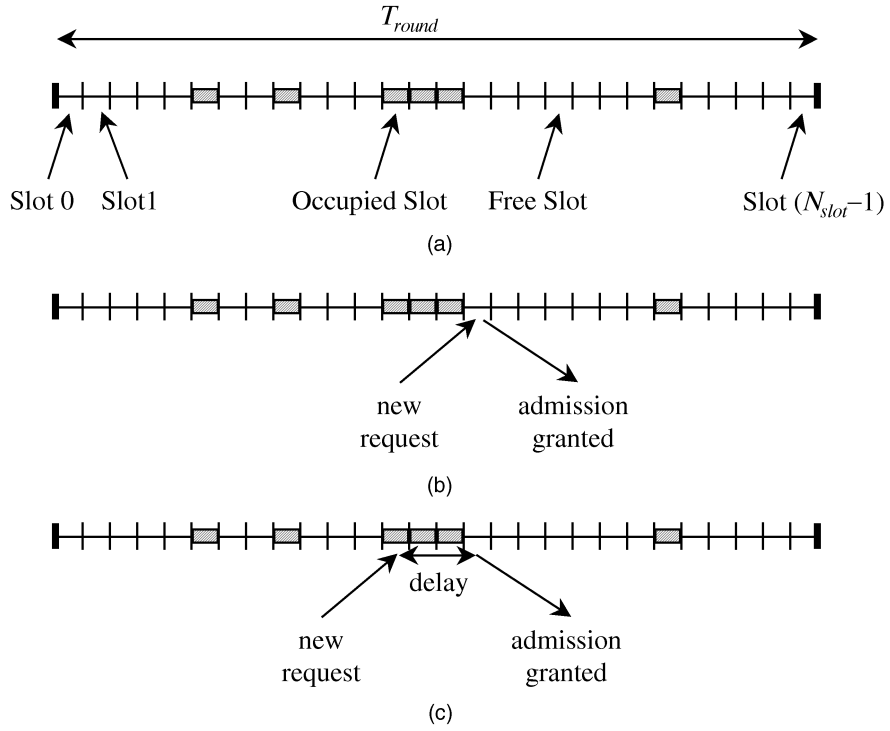

Fig. 6. (a) The admission scheduler with period T_{round} and N_{slot} admission slots. (b) Immediately granting a new video session if the requested slot is free. (c) Delaying a new video session until a free slot is available.

3.1 Start-Time Staggering

We propose a staggering scheme with period length T_{round} and N_{slot} slots as shown in Fig. 6a for the admission scheduler. The period length is determined according to the number of servers in the system as well as the average video playback rate (see Section 4.1). Each time slot has two states: free or occupied. When a client wants to start a new video session, it will first send a request to the scheduler. Ignoring network and processing delays and, assuming the request arrives at the scheduler at time t, the scheduler will admit the new session if and only if the time slot

$$n = \lfloor \mathrm{mod}(t, T_{round})/(T_{round}/N_{slot}) \rfloor \quad (1)$$

is free (Fig. 6b). After admitting the new session (by sending response back to the client), the time slot will be marked as occupied until the session terminates. Conversely, if the requested time slot is occupied, the scheduler will wait (effectively increasing t) until a free time slot is available (Fig. 6c). In this way, the session start times will be staggered in time and, hence, client synchrony reduced. Note that a client only contacts the admission scheduler once at the start and once at the end of a video session. In between, the client sends requests directly to the server nodes. We analyze the effect of this start-time staggering in Section 4.

3.2 Scheduling Delay

The scheduler proposed in the previous section may incur extra scheduling delay to the client. We define scheduling delay as the time from the client requests a new video session to the time the request is granted by the admission scheduler. Assume that n out of the N_{slot} slots are occupied. Then for a client requesting a new video session at an

arbitrary time instance t, the probability of having a free slot immediately is given by

$$V_0 = \frac{N_{slot} - n}{N_{slot}}. \tag{2}$$

Hence, $P_1 = (1 - V_0)$ will be the probability of the requested slot being occupied. Now, provided that the requested slot is occupied, the probability that the next slot is free is

$$V_1 = \Pr\{\text{next slot free} \mid P_1\} = \frac{N_{slot} - n}{N_{slot} - 1}. \tag{3}$$

This is also the probability for a client having to wait one time slot provided that the requested slot is already occupied. It can be shown that the probability for a client to wait k time slots provided that the first k slots are all occupied is

$$V_k = \Pr\{(k+1)\text{th slot free} \mid P_k\} = \frac{N_{slot} - n}{N_{slot} - k} \quad 1 \le k \le n. \tag{4}$$

To obtain P_k, we note that the probability that the first slot is occupied is given by n/N_{slot} and the probability that the second slot is also occupied is given by $(n-1)/(N_{slot}-1)$, and for the third one is $(n-2)/(N_{slot}-2)$, etc. Hence, the probability for the first k slots all occupied is

$$P_k = \prod_{i=0}^{k-1}\left(\frac{n-i}{N_{slot}-i}\right) \qquad 1 \le k \le n. \tag{5}$$

Hence, we can solve for the unconditional probability of a client having to wait k time slots, denoted by W_k, from

$$\begin{aligned} W_k &= \Pr\{(k+1)\text{th slot free} \mid P_k\}P_k \\ &= \left(\frac{N_{slot} - n}{N_{slot} - k}\right)\prod_{i=0}^{k-1}\left(\frac{n-i}{N_{slot}-i}\right) \qquad 1 \le k \le n. \end{aligned} \tag{6}$$

Therefore, given n and N_{slot}, the average and worst-case scheduling delay can be found accordingly. We evaluate this scheduling delay numerically in Section 6.3.

4 Service Delay

In this section, we derive the maximum service delay for the video server to serve a data request sent by a client host. Service delay is defined as the time the server receives the request to the time a complete response (a video data block) is transmitted. We need to know the maximum service delay to determine the amount of client buffer needed to guarantee video playback continuity in Section 5. We first model the request generation process at the video clients in the next section and then proceed to derive the maximum delay incurred in the disk and network subsystems in Sections 4.2 and 4.3, respectively.

4.1 The Request Generation Process

Many studies on VoD system assume that video data are consumed periodically by the video decoder. While this is a reasonable assumption for constant-bit-rate (CBR) video streams, our experience on programming some off-the-shelf hardware and software video decoders nevertheless reveals that the decoder consumes fixed-size data blocks only quasi periodically. This is especially significant in decoders employing interframe compression algorithms like MPEG. Moreover, video titles encoded using constant-quality techniques will also generate variable-bit-rate video streams. In the following, we present a model to account for these variations.

We assume that a request is generated and sent to a server whenever a video block is submitted to the video decoder for playback. Let the average video data rate be R_V and the same for all clients. Since a data block contains Q bytes of video data, the average decoding time for a video block is given by

$$T_{avg} = \frac{Q}{R_V}. \tag{7}$$

To quantify the randomness of video block decoding time, we define a few notations below.

Definition 1.

a. *Given a video stream with average rate R_V, we define T_i as the time the video decoder starts decoding the ith video block. Without loss of generality, we assume decoding starts from time zero, i.e., $T_0 = 0$.*

b. *The decoding-time deviation of video block i is defined as*

$$T_{DV}(i) = T_i - iT_{avg}. \tag{8}$$

Decoding is late if $T_{DV}(i) > 0$ and early if $T_{DV}(i) < 0$.

c. *The maximum lag in decoding is defined as*

$$\begin{aligned} T_L &= \max\{T_{DV}(i) \mid \forall i \ge 0\} \\ &= \max\{T_i - iT_{avg} \mid \forall i \ge 0\}. \end{aligned} \tag{9}$$

d. *The maximum advance in decoding is defined as*

$$\begin{aligned} T_E &= \min\{T_{DV}(i) \mid \forall i \ge 0\} \\ &= \min\{T_i - iT_{avg} \mid \forall i \ge 0\}. \end{aligned} \tag{10}$$

e. *The peak-to-peak decoding-time deviation is defined as*

$$T_{DV} = T_L - T_E. \tag{11}$$

Assuming the bounds T_L and T_E exist, we can prove the following theorem regarding the length of the time between the decoding of any two video blocks.

Theorem 1. *The decoding time t between any two video blocks, i and j ($j > i$), is bounded by*

$$\max\{((j-i)T_{avg} - T_{DV}), 0\} \le t \le ((j-i)T_{avg} + T_{DV}). \tag{12}$$

Proof. Please refer to the Appendix for the proof. □

Since decoding time instances are also request-generation time instances, the previous theorem also bounds the interrequest generation time. Given this bound, we can obtain a lower-bound on the time it takes for a

server to receive k requests from n clients generating requests independently.

Theorem 2. *Assume n clients generating requests independently and each client sends requests to the N_S servers in the system in a round-robin manner; then the minimum time for a server to receive k video data requests is given by*

$$T^{\min}_{Request}(k,n) = \max\left\{\left(\left\lceil\frac{k}{n}\right\rceil - 1\right)N_S T_{avg} - T_{DV}, 0\right\}. \quad (13)$$

Proof. Please refer to the Appendix for the proof. □

From (13), we can see that $T^{\min}_{Request}(k,n)$ decreases for larger values of n. In particular, a server may receive k $(k \leq n)$ requests simultaneously in a time instant (i.e., $T^{\min}_{Request}(k,n) = 0$) if multiple clients happen to be synchronized. As discussed in Section 3, we can avoid this kind of client synchrony by staggering the start times of each video session. Setting $T_{round} = N_S T_{avg}$, we can derive a bound similar to Theorem 2 for the case with admission scheduling in place.

Theorem 3. *If the admission scheduler as described in Section 3 is used with parameters $T_{round} = N_S T_{avg}$ and there are n clients, then the minimum time for a server to receive k video data requests is given by*

$$T^{\min}_{Request}(k,n) = \begin{cases} \max\{\lfloor k/n\rfloor N_S T_{avg} - T_{DV}, 0\}, & \mathrm{mod}(k,n) = 1 \\ \max\left\{(w(N_{slot}-n)+k-2)\frac{N_S T_{avg}}{N_{slot}} - T_{DV}, 0\right\} & \text{otherwise,} \end{cases} \quad (14)$$

where $w = \lceil\frac{k}{n}\rceil - 1$.

Proof. Please refer to the Appendix for the proof. □

Knowing the pattern for request arrivals, we are ready to derive the delay bound at the disk subsystem in the next section.

4.2 Delay in the Disk Subsystem

Many disk scheduling algorithms [10] have been proposed for delay-sensitive applications. Rather than limiting our results to a specific algorithm, we consider a more general disk model with only two assumptions:

Assumption 1. *We assume the minimum time to read a block of Q bytes from the disk, denoted by $T^{\min}_{read}$, is known.*

Assumption 2. *We assume the maximum time to read k blocks of Q bytes from the disk, denoted by $T^{\max}_{read}(k)$, is known and the function is nondecreasing with respect to k.*

The constant $T^{\min}_{read}$ can be calculated from the maximum transfer rate of the disk subsystem plus any processing delay at the disk controller, etc. The function $T^{\max}_{read}(k)$ may be derived according to the disk-scheduling algorithm. Note that the disk subsystem is not limited to a single disk but can contain any number of disks (e.g., a disk array).

To derive the maximum delay an arriving request can experience, we first need to know in the worst-case how many requests can coexist (servicing and queueing) in the disk subsystem simultaneously. Based on Assumption 2, we can make the following definition:

Definition 2. *Define $N_{Disk}(t)$ as the minimum number of requests served by the disk subsystem in a time interval t during a busy period. We can calculate it from $T^{\max}_{read}(k)$:*

$$N_{Disk}(t) = \max\{k \mid T^{\max}_{read}(k) < t, \forall k \geq 0\}. \quad (15)$$

Based on (14), we can define a similar function for maximum number of requests generated:

Definition 3. *Define $N_{Request}(t,n)$ as the maximum number of requests generated in a time interval t by n video clients. We can derive it from (14):*

$$N_{Request}(t,n) = \max\{k \mid T^{\min}_{Request}(k,n) < t, \forall k \geq 0\}. \quad (16)$$

Using (15) and (16), we can obtain the maximum number of requests inside the disk subsystem using the following procedure:

Theorem 4. *The maximum number of requests that can coexist in the disk subsystem is given by*

$$L_D = \max\{N_{Request}(t,n) - N_{Disk}(t) \mid \forall t\}. \quad (17)$$

Proof. Please refer to the Appendix for the proof. □

The intuition behind Theorem 4 is that both the request arrival process and the disk retrieval process have a long-term average rate. In particular, if the system is not overloaded, then

$$\lim_{t\mapsto\infty} N_{Request}(t,n) < \lim_{t\mapsto\infty} N_{Disk}(t). \quad (18)$$

However, for shorter time interval t, burstiness in the processes can cause requests to arrive temporarily faster than the disk subsystem can serve them, hence resulting in request queueing. The procedure in Theorem 4 essentially finds the worst case among these variations.

Equation (17) bounds the maximum number of requests in the system during any busy period. Substituting L_D into $T^{\max}_{read}(k)$, we can obtain the maximum delay for any request to complete service in the disk subsystem:

$$D^{\max}_{disk} = T^{\max}_{read}(L_D). \quad (19)$$

Equation (19) bounds the worst-case delay to receive service in the disk subsystem, including queueing time and service time under requests generated by n video clients.

4.3 Delay in the Network Subsystem

In this section, we derive the maximum delay in the network subsystem, including queueing delay at the shared queue, and service time at the traffic smoother. First of all, we derive an upper bound for the delay a request can experience upon entering the shared queue.

Theorem 5. *For a request that arrives at the shared queue to find $k-1 (k \geq 1)$ requests already in the system, the delay for this newly arrived request to complete service at the traffic smoother is bounded from above by*

$$T^{\max}_{tx}(k) = \begin{cases} \frac{MQ}{C_S} & \text{if } k \leq M \\ \frac{(k+M)Q - MY}{C_S} & \text{otherwise.} \end{cases} \quad (20)$$

Proof. Please refer to the Appendix for the proof. □

Using similar techniques as in the previous section, we can transform (20) into the time domain:

Definition 4. *Define $N_{tx}(t)$ as the minimum number of requests serviced in a time interval of t during a busy period. We can calculate it using (20):*

$$N_{tx}(t) = \max\{k \mid T_{tx}^{\max}(k) < t,\ \forall k \geq 0\}. \tag{21}$$

Now consider the departure process at the disk subsystem. Below, we derive a theorem bounding the maximum number of departed requests in a time interval t.

Theorem 6. *Assume there are n video clients generating video requests simultaneously and the maximum number of requests in the disk subsystem is L_D, then the maximum number of requests departing from the disk subsystem in a time interval t is*

$$N_{out}(t,n) = \min\left\{\left\lceil \frac{t}{T_{read}^{\min}} \right\rceil, L_D + N_{Request}(t,n)\right\}. \tag{22}$$

Proof. Please refer to the Appendix for the proof. □

The departure process as described by (22) is just the arrival process at the network subsystem. Therefore, we can obtain the maximum number of requests in the network subsystem at any time using (21) and (22).

Theorem 7. *The maximum number of requests that can coexist in the network subsystem is given by*

$$L_N = \max\{N_{out}(t,n) - N_{tx}(t) \mid \forall t\}. \tag{23}$$

Proof. The proof is similar to Theorem 4. □

Equation (23) bounds the maximum number of requests in the network subsystem at any time. Substituting L_N into (20), we can obtain the maximum delay for any request to complete service in the network subsystem:

$$D_{\max}^{net} = T_{tx}^{\max}(L_N). \tag{24}$$

Equation (24) bounds the worst-case delay to receive service in the network subsystem, including queueing time and service time.

4.4 Overall System Delay

The previous sections derive the worst-case delay at the disk subsystem and network subsystem, respectively. Depending on the hardware, operating system, and server software implementation, a request may experience additional processing delays, denoted by $D_{\max}^{proc}$. Hence, the worst-case overall service delay at the server is

$$D_{\max} = D_{\max}^{proc} + D_{\max}^{disk} + D_{\max}^{net}. \tag{25}$$

This parameter is useful for the system designer to dimension the amount of buffers required at the client for video continuity, described in the next section.

5 Client Playback Continuity and Buffer Requirement

Buffers are used at the client to absorb service delay and variations in the request generation process. Let N_B be the number of blocks of buffers (each Q bytes) available at the client host, managed as a circular buffer. Before playback begins, the client prefetchs the first $(N_B - 1)$ blocks with video data. After playback starts, the client will generate one request for every video block submitted to the video decoder. Base on this buffer management model, in the next section we derive the relationship between the service delay D_{max} and the client buffer size to guarantee video playback continuity.

5.1 Buffering for Playback Continuity

When playback starts, the first video block is submitted to the decoder at time T_0 (c.f. Definition 1). Consider an arbitrary video block i, the decoding time T_i is bounded by

$$iT_{avg} + T_E \leq T_i \leq iT_{avg} + T_L. \tag{26}$$

Now consider the time instant, denoted by F_i, video block i is received by the client. Since the first $(N_B - 1)$ video blocks are prefetched before playback starts, we have $F_i \leq T_0, \forall i < (N_B - 1)$. Ignoring network delay, the filling time for the remaining video blocks can be calculated from the request-generation time and the maximum service delay as follows:

$$F_i \leq \max\{T_{i-N_B+1}\} + D_{\max} \qquad \forall i \geq (N_B - 1). \tag{27}$$

From Theorem 1, we can simplify $\max\{T_i - N_B + 1\}$ to obtain

$$F_i \leq (i - N_B + 1)T_{avg} + T_L + D_{\max} \qquad \forall i \geq (N_B - 1). \tag{28}$$

To ensure playback continuity, the latest filling time of a video block must be smaller than the earliest decoding time:

$$F_i \leq T_i \qquad \forall i. \tag{29}$$

Substituting (26) and (28) into (29), we can then obtain the relation between $D_{\max}$ and N_B:

$$(i - N_B + 1)T_{avg} + T_L + D_{\max} \leq iT_{avg} + T_E. \tag{30}$$

Noting that $T_{DV} = T_L - T_E$, we can rearrange (30) to obtain

$$N_B \geq \frac{D_{\max} + T_{DV}}{T_{avg}} + 1. \tag{31}$$

This formula allows us to calculate the amount of client buffers needed to ensure playback continuity once the other parameters ($D_{max}, T_{DV},$ and T_{avg}) are known.

5.2 Prefetch Delay

As the first $(N_B - 1)$ video buffers must be prefetched before playback starts, an additional prefetch delay is incurred. This delay is needed whenever the client buffer must be cleared and then reprefetched, such as seeking within the video stream. Other interactive controls like pause/resume, slow, and frame stepping are not affected. We assume that prefetch requests are generated periodically

TABLE 1
System Parameters for Performance Evaluation

System Parameters	Symbol	Value
Spindle speed	n/a	5411 rpm
Max latency	$T_{latency}$	11ms
Number of tracks	N_{track}	2621
Single-track seek	n/a	1ms
Average seek	n/a	10ms
Max full-stroke seek	n/a	19ms
Video packet size	Y	8192 Bytes
Video block size	Q	65536 Bytes
Video data rate	R_V	150KB/s
Number of send queues in traffic smoother	M	10
Raw disk transfer rate (Seagate ST12400N)	R_{disk}	3.35MB/s
Average video block decoding time	T_{avg}	437ms
Maximum advance in decoding time	T_E	-130ms
Maximum lag in decoding time	T_L	160ms
Effective network throughput	C_S	1.875MB/s
Maximum processing delay	$D_{\max}^{proc}$	0ms

with an interval of T_{avg}. Then the maximum time to complete the prefetch process can be calculated from

$$D_{Prefetch} = (N_B - 2)T_{avg} + D_{\max}. \quad (32)$$

The prefetch delay is an important design parameter as it directly affects the responsiveness of the system. By rearranging (32), we can calculate the maximum allowed buffer size for a given maximum tolerable prefetch delay.

6 Performance Evaluation

In this section, we present numerical results computed using the performance model in Section 4 and 5 to evaluate various performance metrics. For the general disk model in Section 4.2, we need specific definitions for Assumption 1 and 2 in order to compute numerical results. For simplicity, we assume a single-disk with Circular-SCAN [10] disk scheduling. The disk model and the definitions for the two assumptions are defined in Appendix A.7. The disk parameters and other system parameters are listed in Table 1. Interested readers are referred to [11], [12], [13], [14] for more sophisticated disk models and disk-scheduling algorithms.

6.1 Service Delay versus Number of Concurrent Clients

Fig. 7 shows the service delay versus number of concurrent clients for an eight-server system. Note that the discrete jumps in the graph are due to the round-based scheduling being used in the disk subsystem. As requests may be served out-of-order within a round, the worst-case delay will jump according to how many service rounds are required to serve a given number of requests.

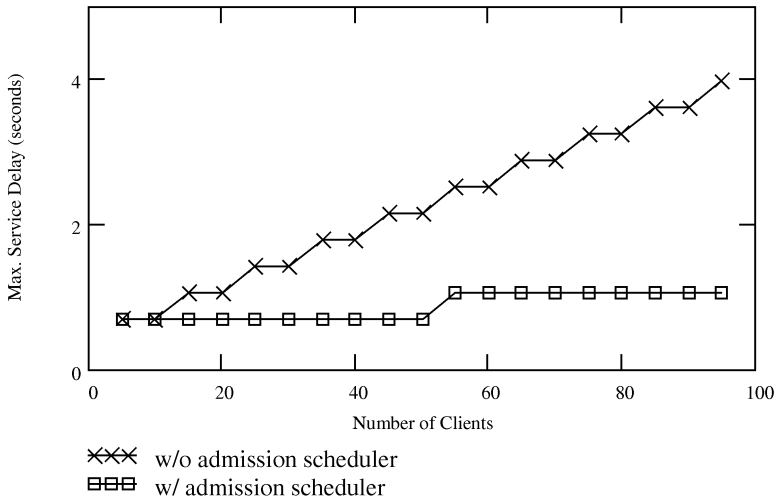

Fig. 7. Maximum service delay versus number of current clients for an eight-servers system.

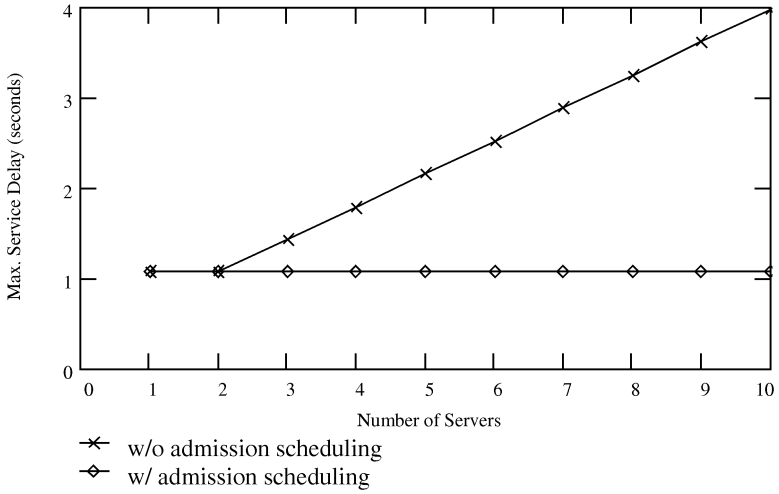

Fig. 8. Maximum service delay versus number of servers for a client-server ratio of 10.

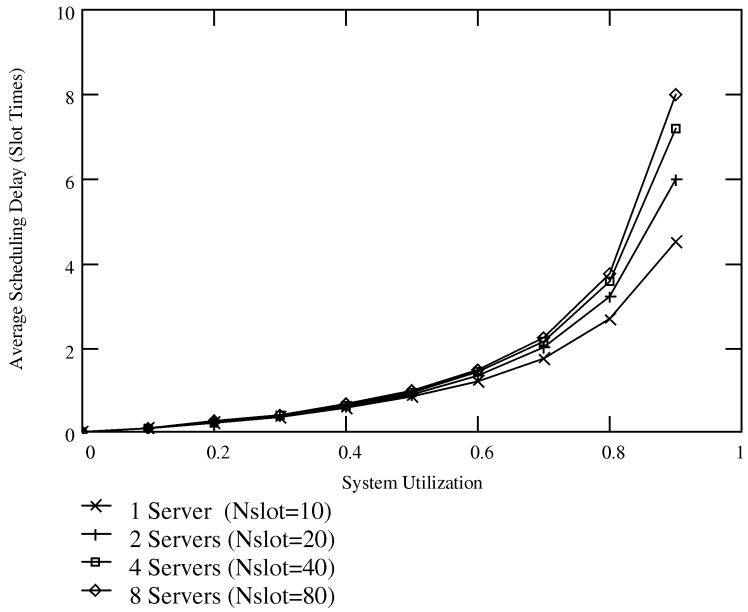

Fig. 9. Average scheduling delay versus system utilization.

Note that without the admission scheduler, the service delay increases with the number of concurrent clients at a much higher rate. This is a direct result of the client synchrony problem described and modeled in Section 4.1. Conversely, the maximum service delay stays relatively low if the admission scheduler is employed. This shows that by staggering the request-generation times, the admission scheduler can effectively avoid client synchrony.

On the other hand, it may appear that the system is lightly loaded as each video session requires only 150KB/s while the disk has a raw throughput of 3.35MB/s (Table 1). However, the effective disk throughput can never approach the raw throughput in practice due to seek-time overhead, rotational latency, etc. For example, using C-SCAN with a round size of 10. The disk in Table 1 (Seagate ST12400N) can theoretically sustain at most 12 video sessions (c.f. Appendix A.7). Indeed, Theorem 4 diverges (i.e., $D_{\max} \to \infty$) for more than 96 clients, indicating a maximum client-server ratio of 96/8=12.

6.2 Service Delay versus Number of Servers

Fig. 8 shows the service delay versus number of servers in the system with a client-server ratio of 10. Note that without the admission scheduler, the service delay increases linearly with the number of servers. Conversely, the service delay remains constant regardless of the number of servers as long as the client-server ratio is also constant. This strongly suggests that the system can be scaled up linearly if the admission scheduler is employed. We will study this scalability issue in Section 6.5.

6.3 Average Scheduling Delay versus System Utilization

The price to pay for using the admission scheduler is extra scheduling delay at the start of a new video session. For a given system utilization (n/N_{slot}) where n is the number of concurrent video sessions, the average scheduling delay can be calculated from (6). Assuming a client-server ratio of $R_{CS} = 10$, we plot the average scheduling delay for various number of servers in Fig. 9. While the worst-case scheduling

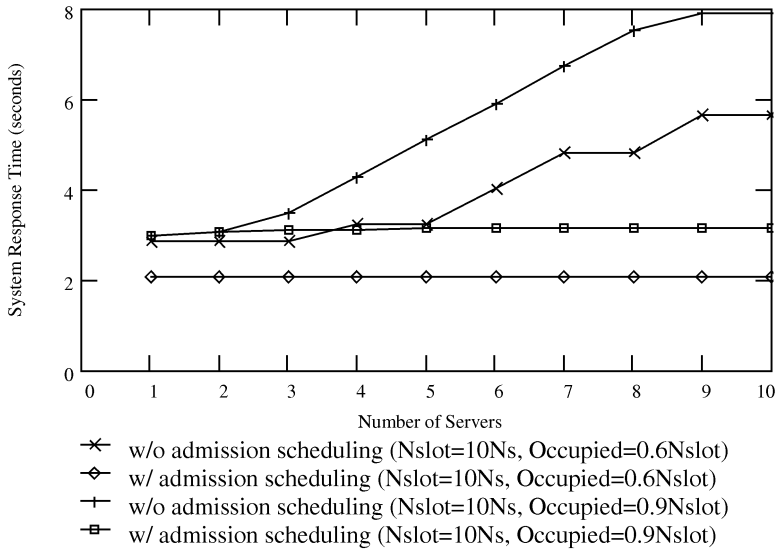

Fig. 10. System response time versus number of servers.

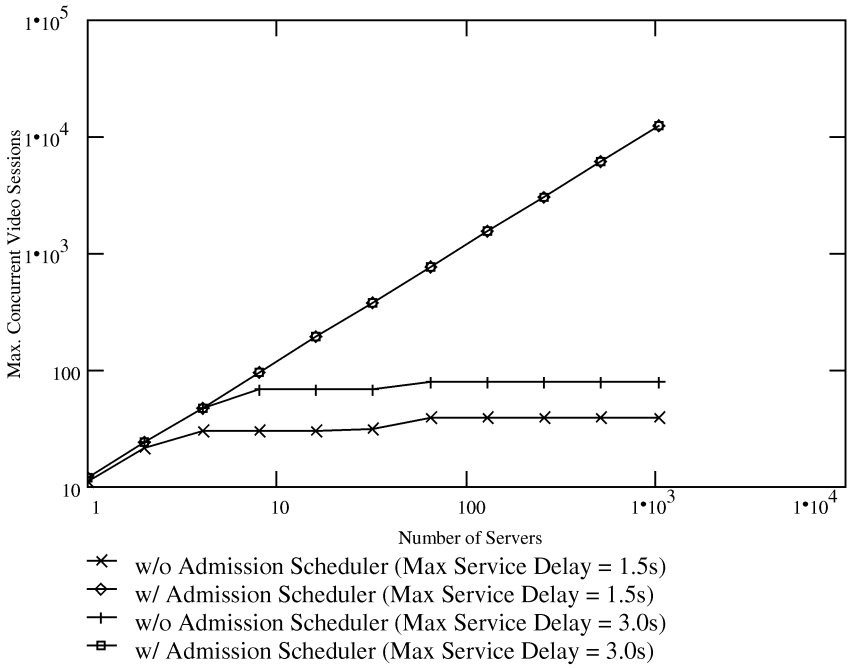

Fig. 11. Maximum number of concurrent video sessions versus number of servers.

delay (precisely $R_{CS}N_S - 1$ time slots) increases for more servers, the average scheduling delay is significantly smaller for system utilization as high as 0.9 (e.g., less than 10 time slots for eight servers at 0.9 utilization, versus the worst-case of 79 time slots). Hence, by limiting the maximum system utilization (such as 0.9), we can reduce the scheduling delay to reasonable ranges.

6.4 System Response Time versus Number of Servers

The previous sections have shown that while the admission scheduler can reduce the service delay, it also introduces extra scheduling delay during admission. Therefore, we need to compare the overall system response time, defined as the maximum prefetch delay plus the scheduling delay, to better evaluate the effect of the admission scheduler on the responsiveness of the system. Fig. 10 plots the system response time versus number of servers for system utilization (i.e., n/N_{slot}) of 0.6 and 0.9, respectively. The results show that even with the addition of extra scheduling delay, the total system response time is still smaller when admission scheduling is employed.

6.5 Scalability Under Delay Constraint

To evaluate the scalability of the parallel video server studied in this paper, we set a limit on the maximum service delay and compute the maximum number of concurrent clients that can be served. By constraining the service delay, we effectively fixed the client buffer requirement (Section 5.1) and the resulting prefetch delay (Section 5.2) incurred. The results are summarized in Fig. 11 for system size ranging from one to 1,024 servers. Note that

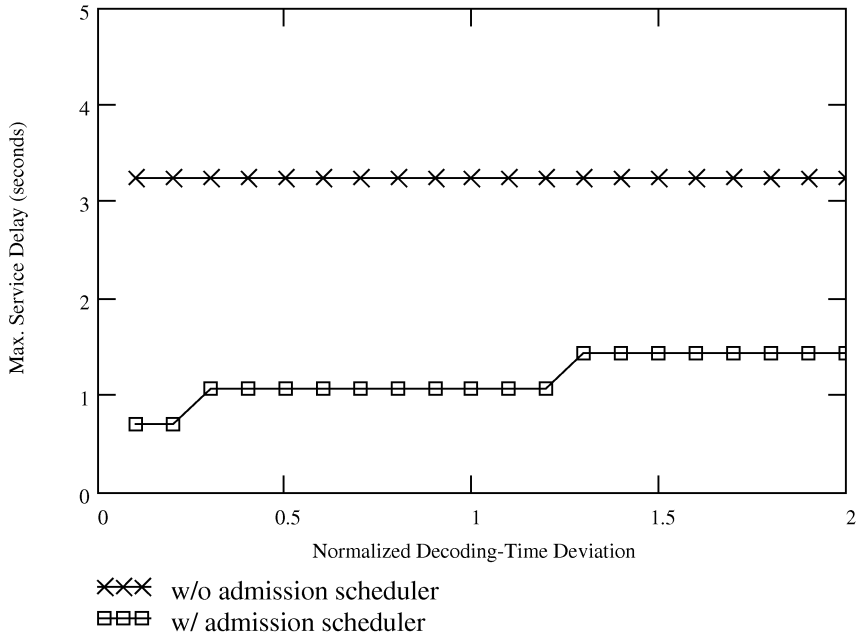

Fig. 12. Maximum service delay versus normalized decoding-time deviation.

without the admission scheduler, the aggregate capacity levels off for more than eight servers due to the client synchrony problem. Conversely, the system is linearly scalable if the proposed admission scheduler is employed. Note that the two curves for maximum service delays of 1.5 seconds and three seconds (with admission scheduling) overlap. This is because the service delay has discrete jumps as shown in Fig. 7 and in this case, exceeds three seconds in the next jump. Hence, the system capacity stays the same even though the service delay limit is relaxed from 1.5 seconds to three seconds.

6.6 Sensitivity to Decoding-Time Deviation

The decoding time deviations T_E and T_L in Table 1 are measured experimentally using an off-the-shelf hardware MPEG-1 decoder over many video streams. To evaluate the sensitivity of the system to this variable, we plot the maximum service delay versus a normalized peak-to-peak decoding time deviation in Fig. 12 for eight servers and 80 clients. The normalization is obtained from dividing the deviation T_{DV} by the average decoding time T_{avg}. Surprisingly, the curve for no admission scheduler remains invariant to increases in T_{DV}. For the case with admission scheduling, the delay increases slightly for very large deviations. For example, the delay is increased only by 33.8 percent for a T_{DV} equivalent to 200 percent the value of T_{avg}. This shows that the system is not sensitive to decoding-time deviations.

7 Related Works

The principle of using parallel devices to achieve scalability has been studied in various contexts. For example, parallelism has been proposed for disk arrays [15], tape arrays [16], [17], and even network transmissions [18]. In particular, the use of parallel disk systems have been studied extensively in VoD systems literatures [11], [12], [13], [14]. Recently, there is an increasing interest in exploiting server-level parallelism for designing scalable VoD systems. For example, studies have been conducted by[1] Biersack et al. [19], [20], Bolosky et al. [21], Buddhikot, and Parulkar [22], Freedman and DeWitt [23], Ghandeharizadeh and Ramos [24], Lee and Wong [7], [9], Lougher et al. [25], Reddy [26], Tewari et al. [27], and Wu and Shu [28]. A comprehensive study of architectural alternatives and the approaches employed by existing systems can be found in [29]. Below, we highlight and compare the key differences between the mentioned studies and the architecture studied in this paper.

The pioneering study by Ghandeharizadeh and Ramos [11], [24] considered the retrieval of multimedia data from a parallel multimedia information system. They proposed striping multimedia data over multiple disks located in storage nodes connected by a high-speed network. Their study focused on disk I/O bottleneck and stripping policy designed to support multimedia data objects with different playback bit-rates. Their striping algorithm also requires the scheduling of arriving requests to prevent disk load imbalance. This paper differs from their studies in two major ways: (a) our study incorporated issues in network transmission, bottleneck in CPU processing, and synchronization among servers, which are not considered in [11], [24]; and (b) we develop our admission scheduler for client-pull service model rather than server-push service model.

The studies by Reddy [26], Tewari et al. [27], and Wu et al. [28] are based on architectures where one or more intermediate delivery nodes are used to merge video data from multiple servers for delivery to clients. Conversely, in our architecture, a video client retrieves video data directly from multiple servers without passing through an intermediate node. Our approach eliminates the extra hardware

1. In alphabetical order.

needed to run the intermediate delivery nodes, which themselves could become a bottleneck of the system.

Additionally, the architecture studied in this paper employs the client-pull service model rather than the common server-push service model as employed in [19], [21], [22], [26], [27], [28]. This model eliminates the need for interserver synchronization, which is required for the server-push service model. Biersack et al. [20] have studied the problem for server-push designs and proposed algorithms to compensate network delay differences and clock drifts. Other researchers like Buddhikot et al. [22] solve the problem by using closely-coupled parallel servers having hardware-synchronized clocks instead of loosely-coupled servers running independently of each other. The study by Bolosky et al. [21] employs a separate controller to synchronize the clocks in the servers so that transmissions from the push-based servers are properly coordinated. The controller is linked up with the servers by a separate network to minimize network latency and to avoid interference from video traffic. On the contrary, no such clock-synchronization is required for the client-pull service model. Another study by Rao et al. [30] presented qualitative as well as quantitative (through simulation) comparisons between the two service models. However, their study is for single-server systems only and, hence, did not address the instantaneous load imbalance problem tackled in this paper.

Another difference is in performance analysis. The work by Biersack et al. [19], [20] focuses on synchronization issues while those by Buddhikot et al. [22] and Wu et al. [28] focus on data placement, scheduling, and playout control. Our study focuses on the performance of the system in relation to various system parameters. We study the scalability of the system under the constraint that video continuity is guaranteed. Base on a few assumptions, we modeled the system performance using worst-case analysis and computed numerical results for performance evaluation. In [27], Tewari et al. have also studied system performance and scalability. They apply queueing analysis and model the system using results from M/D/1 queue. However, their analysis does not consider the video clients and focuses on the back-end storage nodes and the delivery nodes only. They did not consider admission scheduling and assumed Poisson request arrivals.

Finally, our proposal on the use of an admission scheduler to control instantaneous load imbalance in the pull-based parallel video server is also new. Our results showed that the admission scheduler is crucial to achieving linear scalability in a pull-based parallel video server.

8 Conclusion

In this paper, we have analyzed and evaluated the performance of a pull-based parallel video server architecture. The architecture employs striping at the server level for perfect load sharing across multiple autonomous servers. A client-pull service model is employed to control the delivery of video data from multiple servers to a client in the absence of interserver synchronization. Our study on the aggregate request-generation process of multiple clients reveals that client synchrony could lead to instantaneous server overload. To solve this problem, we proposed and analyzed a staggering-based admission scheduler. By modeling the server's service delay and the client buffer requirement, we showed (via numerical results) that with the admission scheduler employed, the parallel video server architecture can be scaled up linearly simply by adding more servers and redistributing the data among them.

There are still many areas in parallel video server design that warrant more investigations. For example, we have ignored network and processing delay in our analysis. While network delay in a LAN environment is insignificant compared to the service delay, the case in WAN environment may be significant. Secondly, some early studies [9], [21] have demonstrated that a parallel video server is not only scalable, but can also be made fault tolerant by introducing data redundancy among the servers. More studies are needed to characterize the performance of a parallel video server under server failures.

Appendix

A.1 Proof of Theorem 1

Lower bound:

$$\begin{aligned} T_j - T_i &= \big(jT_{avg} + T_{DV}(j)\big) - \big(iT_{avg} + T_{DV}(i)\big) \\ &\geq (j-i)T_{avg} + T_E - T_L \\ &= (j-i)T_{avg} - T_{DV}. \end{aligned} \tag{A.1}$$

The proof for upper bound is similar. □

A.2 Proof of Theorem 2

We denote the jth request generated by client i by $r_{i,j}$ and let $T_{i,j}$ be the time the request is generated. As a client distributes requests among N_S servers in a round-robin manner, two requests sending to the same server will be separated by $(N_S - 1)$ other requests. Hence, the minimum time for a server to receive m requests from an arbitrary client, denoted by $T^{\min}_{Single}(m)$, can be calculated from (c.f. Theorem 1):

$$T^{\min}_{Single}(m) = (m-1)N_S T_{avg} - T_{DV}. \tag{A.2}$$

The above equation is applicable to any one of the n clients generating requests simultaneously. Therefore, if a server can receive a maximum of m requests from any one client in a time interval t, then this also implies that the server can receive a maximum of m requests from each one of the i clients. Hence, we can determine the minimum time for a server to receive k requests from (A.2) as follows:

$$\begin{aligned} T^{\min}_{Request}(k,n) &= T^{\min}_{Single}\left(\left\lceil \frac{k}{n} \right\rceil\right) \\ &= \left(\left\lceil \frac{k}{n} \right\rceil - 1\right) N_S T_{avg} - T_{DV}. \end{aligned} \tag{A.3}$$

Obviously, the minimum time must be greater than or equal to zero and the result follows. □

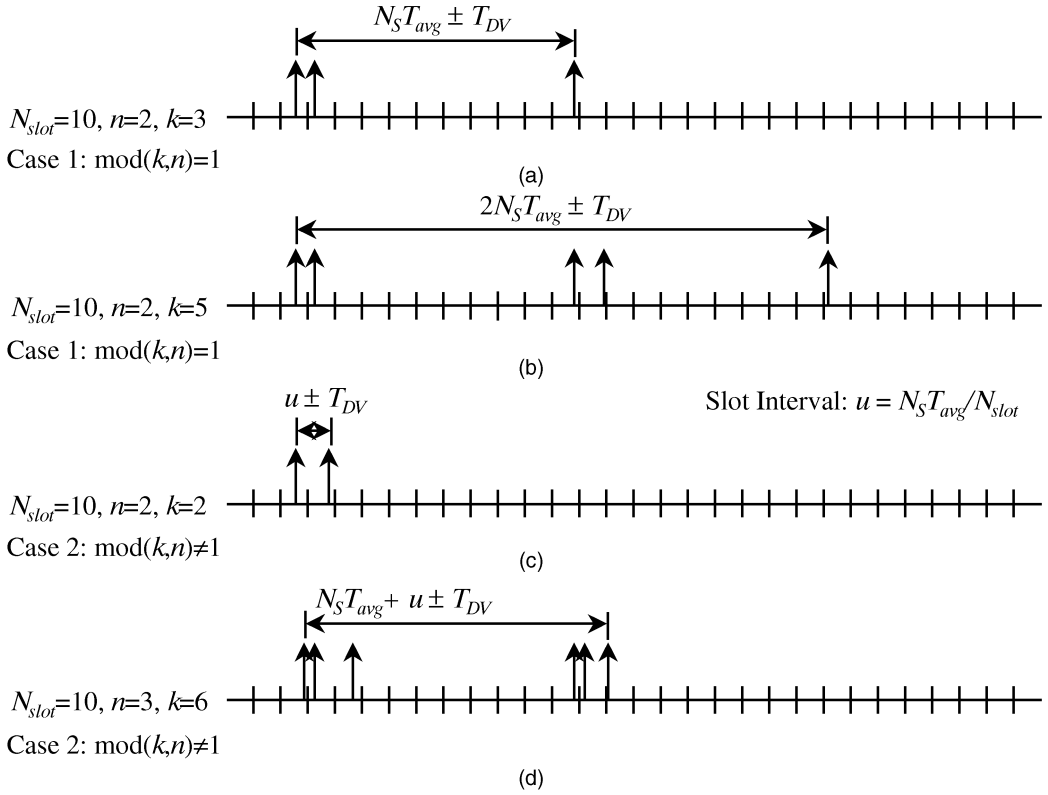

Fig. 13. Worst-case scenarios for requests arriving at the server.

A.3 Proof of Theorem 3

As a client sending requests to N_S servers in a round-robin manner, a server will receive requests from the client with an average interval of $N_S T_{avg}$. Let $T_{i,j}$ be the time for a server to receive the jth request from client i. Then we prove the two cases separately below.

Case 1 ($\mathrm{mod}(k,n) = 1$.). In this case, the first and the last requests come from the same client (see Fig. 13a and 13b). Let the first request arrives at time $T_{i,j}$. Then the last request arrives at time $T_{i,w}$, where $w = \lfloor k/n \rfloor$. Hence, we can determine the minimum time interval from Theorem 1 directly

$$\begin{aligned} T_{i,j+w} - T_{i,j} &\leq wN_S T_{avg} - T_{DV} \\ &= \lfloor k/n \rfloor N_S T_{avg} - T_{DV}. \end{aligned} \tag{A.4}$$

Case 2. ($\mathrm{mod}(k,n) \neq 1$.). In this case, the first and the last requests do not come from the same client (see Fig. 13c and 13d). Let the first request arrive at time $T_{i,j}$ and the last request arrive at time $T_{x,y}$. Since there are in total k requests and n clients, each client should have generated at least $w = \lceil k/n \rceil - 1$ requests. On average, the time between $T_{i,j}$ and $T_{i,j+w-1}$ will be $wN_S T_{avg}$. For the remaining $z = (k - wn)$ requests, the minimum time span is just $((z-1-1)(N_S T_{avg}/N_{slot}))$. We subtract one from $(z-1)$ to cater for differences in start times of different clients. Hence, the total time interval is

$$T = wN_S T_{avg} + (k - wn - 2)(N_S T_{avg}/N_{slot}). \tag{A.5}$$

From Theorem 1, the lower bound is

$$\begin{aligned} T^{\min}_{Request}(k,n) &= wN_S T_{avg} + (k - wn - 2)(N_S T_{avg}/N_{slot}) - T_{DV} \\ &= (wN_{slot} + k - wn - 2)\frac{N_S T_{avg}}{N_{slot}} - T_{DV} \\ &= (w(N_{slot} - n) + k - 2)\frac{N_S T_{avg}}{N_{slot}} - T_{DV}. \end{aligned} \tag{A.6}$$

Together with the fact that $T^{\min}_{Request}(k,n)$ must be zero or larger and the results follow. □

A.4 Proof of Theorem 4

We prove this theorem by contradiction. Consider an arbitrary busy period in the disk subsystem. Let t_i, $(i > 0)$ be the arrival time of the ith request of the busy period. Assume that the queue length, as observed by the ith arriving request, be $(L-1)$, where $L > L_D$. Then from the start of the busy period to time t_i exactly i requests will have arrived. Since there are L requests (including request i) left in the system at time t_i, $(i - L)$ requests must have completed service. From (15), we know that $(i - L) \geq N_{Disk}(t_i - t_1)$. On the other hand, from (16), we know that $i \leq N_{Request}(t_i - t_1, n)$. Combining, we then have

$$\begin{aligned} (i - L) &\geq N_{Disk}(t_i - t_1) \\ L &\leq i - N_{Disk}(t_i - t_1) \\ &\leq N_{Request}(t_i - t_1, n) - N_{Disk}(t_i - t_1) \\ &\leq L_D, \end{aligned} \tag{A.7}$$

which contradicts with the $L > L_D$ assumption and the result follows. □

A.5 Proof of Theorem 5

Let t_a be the time instance request i enters the shared queue, t_b be the time instance it enters one of the send queue. Then we consider two cases below.

Case 1 ($k \leq M$.). In this case, request i directly enters into one of the send queues without queueing delay, i.e., $t_a = t_b$. In the worst case, the set of requests $\{j \mid i < j \leq (i + M - k)\}$ will arrive at the same time instance t_a. Thereby filling up all free send queues at the traffic smoother. Therefore, the maximum service time for request i is

$$T_{tx}^{\max}(k) = \frac{MQ}{C_S}. \tag{A.8}$$

Case 2 ($k > M$.). In this case, request i will experience queueing delay at the shared queue, i.e., $t_b > t_a$. We first consider the queueing time $(t_b - t_a)$. Let $c_i(t)$ be the amount of data of request i that have not been transmitted at time t. Then at the time request i enters into the shared queue, the total amount of data in the network subsystem is given by

$$U(t_a) = \sum_{x=i-k+1}^{i} c_x(t_a). \tag{A.9}$$

Similarly, at the time request i enters one of the send queues, the total amount of data in the traffic smoother can be obtained from

$$V(t_b) = \sum_{y=i-M+1}^{i} c_y(t_b). \tag{A.10}$$

Therefore, the amount of data transmitted by the traffic smoother during request i's queueing time is just the difference of (A.9) and (A.10):

$$U(t_a) - V(t_b) = \sum_{x=i-k+1}^{i} c_x(t_a) - \sum_{y=i-M+1}^{i} c_y(t_b). \tag{A.11}$$

For the traffic smoother, the time needed to deliver these data can be calculated from

$$\begin{aligned} t_b - t_a &= \frac{U(t_a) - V(t_b)}{C_S} \\ &= \frac{1}{C_S}\left(\sum_{x=i-k+1}^{i} c_x(t_a) - \sum_{y=i-M+1}^{i} c_y(t_b)\right) \\ &\leq \frac{1}{C_S}\left(\sum_{x=i-k+1}^{i} Q - \sum_{y=i-M+1}^{i} c_y(t_b)\right) \quad since\ c_x(t_a) \leq Q \\ &\leq \frac{1}{C_S}\left(\sum_{x=i-k+1}^{i} Q - \sum_{y=i-M+1}^{i} Y\right) \quad since\ c_y(t_b) \geq Y \\ &= \frac{kQ - MY}{C_S}. \end{aligned} \tag{A.12}$$

After the queueing delay, request i will enter one of the send queues for transmission. The maximum service time is the same as that in Case 1. Therefore, the maximum total time delay is just the sum of (A.8) and (A.12):

$$T_{tx}^{\max}(k) = \frac{kQ - MY}{C_S} + \frac{MQ}{C_S} \tag{A.13}$$

and the result follows. □

A.6 Proof of Theorem 6

First, the minimum time between two departing requests from the disk subsystem is bounded by $T_{read}^{\min}$. Hence, the minimum interdeparture time is simply $T_{read}^{\min}$.

From Theorem 5, we know that there are at most L_D requests in the disk subsystem. In the worst-case, the disk can service requests at the peak rate as long as there is requests in the disk subsystem. Since $N_{Request}(t, n)$ is the maximum number of request that can arrive at the disk subsystem in a time interval t, the disk will run out of requests to service if $t > (L_D + N_{Request}(t, n))T_{read}^{\min}$. This proves the first part of Theorem 6. The second part follows from the observation that at any later time t, the maximum possible number of requests serviced by the disk subsystem is just the total number of requests arrived during this time interval—$N_{Request}(t, n)$, plus the L_D requests already in the system. □

A.7 Disk Model

We assume the disk has N_{track} tracks and the disk seek function is given by [31]

$$T_{seek}(n) = \alpha n + \beta\sqrt{n} + \gamma. \tag{A.14}$$

The constants α, β, and γ can be determined from the track-to-tack seek time, average seek time, and full-stroke seek time of the disk drive. Let R_{disk} be the raw disk transfer rate, then the minimum time to read a block of Q bytes from the disk is (c.f. Assumption 1):

$$T_{read}^{\min} = T_{seek}(0) + \frac{Q}{R_{disk}}. \tag{A.15}$$

Let $T_{latency}$ be the maximum rotational latency, then the time to seek n tracks and read a block of Q bytes is

$$T_{read}(n) = \alpha n + \beta\sqrt{n} + \gamma + T_{latency} + \frac{Q}{R_{disk}}. \tag{A.16}$$

It can be shown that the worst-case round time for C-SCAN to service k requests is given by

$$\begin{aligned} T_{cscan}(k) = (k+1)&\left(\alpha\frac{N_{track}}{k+1} + \beta\sqrt{\frac{N_{track}}{k+1}} + \gamma\right) \\ &+ k\left(T_{latency} + \frac{Q}{R_{disk}}\right), \end{aligned} \tag{A.17}$$

where the k requests are evenly distributed across the disk surface. As the C-SCAN scheduling algorithm may service requests out of order in a round, we can only guarantee that a request will be completed at the end of a service round. Assume that the disk will serve at most N_{round} requests in a round, then the maximum time to read k requests off the disk with C-SCAN is given by (c.f. Assumption 2):

$$T_{read}^{\max}(k) = \left\lceil \frac{k}{N_{round}} \right\rceil T_{\text{csc } an}(N_{round}). \quad \text{(A.18)}$$

Note that with C-SCAN, it is possible to obtain a theoretical upper bound for the number of concurrent video sessions that can be supported using the traditional server-push architecture. For example, assuming one video block of Q bytes is retrieved for each video stream in a service round, then the worst-case length of a service round must be shorter than the average playback duration for a video block of Q bytes. Hence, the upper bound can be determined by finding the maximum k such that $T_{CSCAN}(k) \leq T_{avg}$. For the Seagate ST12400N hard disk used in Section 6, up to 12 concurrent video sessions can be supported using C-SCAN. □

ACKNOWLEDGMENTS

This research was funded in part by a research grant (CUHK6095/99E) from the HKSAR Research Grants Council and in part by the Area of Excellence in IT—a research grant from the HKSAR University Grants Council. The authors would like to thank the anonymous reviewers for their constructive comments and suggestions in improving this paper to its final form.

REFERENCES

[1] R.L. Haskin, "The Shark Continuous-Media File Server," *Proc. COMPCON '93,* pp. 12-15, 1993.
[2] F.A. Tobagi and J. Pang, "StarWorks—A Video Applications Server," *Proc. IEEE COMPCON '93,* pp. 4-11, 1993.
[3] H. Taylor, D. Chin, and S. Knight, "The Magic Video-on-Demand Server and Real-Time Simulation System," *IEEE Parallel and Distributed Technology: Systems and Applications,* vol. 3, no. 2, pp. 40-51, 1995.
[4] R. Buck, "The Oracle Media Server for nCube Massively Parallel Systems," *Proc. Eighth Int'l Parallel Processing Symp.,* pp. 670-673, 1994.
[5] S.A. Barnett and G.J. Anido, "A Cost Comparison of Distributed and Centralized Approaches to Video-on-Demand," *IEEE J. Selected Areas on Comm.,* vol. 14, no. 6, pp. 1,173-1,183, 1996.
[6] C.C. Bisdikian and B.V. Patel, "Issues on Movie Allocation in Distributed Video-on-Demand Systems," *Proc. Int'l Computer Counsel '95,* pp. 250-255, 1995.
[7] Y.B. Lee and P.C. Wong, "A Server Array Approach for Video-on-Demand Service on Local Area Networks," *Proc. IEEE INFOCOM '96,* Mar. 1996.
[8] D.A. Patterson, P. Chen, G. Gibson, and R.H. Katz, "Introduction to Redundanct Array of Inexpensive Disks (RAID)," *Proc. COMPCON '89,* pp. 112-117, 1989.
[9] P.C. Wong and Y.B. Lee, "Redundant Array of Inexpensive Servers (RAIS) for On-Demand Multimedia Services," *Proc. ICC '97 Workshop Information Infrastructure,* June 1997.
[10] A.L.N. Reddy and J.C. Wyllie, "I/O Issues in a Multimedia System," *Computer,* vol. 27, no. 3, pp. 69-74, Mar. 1994.
[11] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, "Staggered Striping in Multimedia Information Systems," *Proc. SIGMOD '94,* pp. 79-90, May 1994.
[12] A.N. Mourad, "Issues in the Design of a Storage Server for Video-on-Demand," *ACM Multimedia Systems,* vol. 4, pp. 70-86, 1996.
[13] P.S. Yu, M.S. Chen, and D.D. Kandlur, "Grouped Sweeping Scheduling for DASD-Based Multimedia Storage Management," *ACM Multimedia Systems,* vol. 1, pp. 99-109, 1993.
[14] F.A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID®—A Disk Array Management System for Video Files," *Proc. ACM Multimedia,* pp. 393-400, 1993.
[15] K. Salem and H. Garcia-Molina, "Disk Striping," *Proc. IEEE Int'l Conf. Data Engineering,* 1986.
[16] A.L. Drapeau and R.H. Katz, "Striped Tape Arrays," *Proc. 12th IEEE Symp. Mass Storage Systems,* pp. 257-265, 1993.
[17] L. Golubchik, R.R. Muntz, and R.W. Watson, "Analysis of Striping Techniques in Robotic Storage Libraries," *Proc. 14th IEEE Symp. Mass Storage Systems,* pp. 225-238, Sept. 1995.
[18] C. Brendan, S. Traw, and J.M. Smith, "Striping within the Network Subsystem," *IEEE Network,* pp. 22-32, July/Aug. 1995.
[19] C. Bernhardt and E. Biersack, "The Server Array: A Scalable Video Server Architecture," *High-Speed Networks for Multimedia Applications,* 1996.
[20] E. Biersack, W. Geyer, and C. Bernhardt, "Intra- and Inter-Stream Synchronization for Stored Multimedia Streams," *Proc. IEEE Int'l Conf. Multimedia Computing & Systems,* June 1996.
[21] W.J. Bolosky, J.S. Barrera III, R.P. Draves, R.P. Fitzgerald, G.A. Gibson, M.B. Jones, S.P. Levi, N.P. Myhrvold, and R.F. Rashid, "The Tiger Video Fileserver," *Proc. Sixth Int'l Workshop Network and Operating System Support for Digital Audio and Video,* Apr. 1996.
[22] M.M. Buddhikot and G.M. Parulkar, "Efficient Data Layout, Scheduling and Playout Control in MARS," *Proc. Fifth Int'l Workshop Network and Operating System Support for Digital Audio and Video,* 1995.
[23] C.S. Freedman and D.J. DeWitt, "The SPIFFI Scalable Video-on-Demand System," *Proc. ACM SIGMOD '95,* June 1995.
[24] S. Ghandeharizadeh and L. Ramos, "Continuous Retrieval of Multimedia Data Using Parallelism," *IEEE Trans. Knowledge and Data Eng.,* vol. 5, no. 4, pp. 658-669, Aug. 1993.
[25] P. Lougher, D. Pegler, and D. Shepherd, "Scalable Storage Servers for Digital Audio and Video," *Proc. IEEE Int'l Conf. Storage and Recording Systems,* pp. 140-143, Apr. 1994.
[26] A. Reddy, "Scheduling and Data Distribution in a Multiprocessor Video Server," *Proc. Second IEEE Int'l Conf. Multimedia Computing and Systems,* 1995.
[27] R. Tewari, R. Mukherjee, and D.M. Dias, "Real-Time Issues for Clustered Multimedia Servers," IBM Research Report RC20020, June 1995.
[28] M. Wu and W. Shu, "Scheduling for Large-Scale Parallel Video Servers," *Proc. Sixth Symp. Frontiers of Massively Parallel Computation,* pp. 126-133, Oct. 1996.
[29] Y.B. Lee, "Parallel Video Servers—A Tutorial," *IEEE Multimedia,* vol. 5, no. 2, pp. 20-28, June 1998.
[30] S.S. Rao, H.M. Vin, and A. Tarafdar, "Comparative Evaluation of Server-Push and Client-Pull Architectures for Multimedia Servers," *Proc. Sixth Int'l Workshop Network and Operating System Support for Digital Audio and Video,* pp. 45-48, Apr. 1996.
[31] M. Tanner, *Practical Queueing Analysis.* McGraw Hill, 1995.

Jack Yiu-Bun Lee received his BEng and PhD degrees from the Department of Information Engineering at the Chinese University of Hong Kong in 1993 and 1997, respectively. From 1998 to 1999, he was assistant professor at the Department of Computer Science at the Hong Kong University of Science and Technology. He is currently assistant professor at the Department of Information Engineering at the Chinese University of Hong Kong. His research interests include distributed multimedia-on-demand systems, fault-tolerant systems, and Internet computing. Dr. Lee is a member of the IEEE and ACM.

Po-Choi Wong has been a lecturer in the Department of Computing Studies of Hong Kong Polytechnic for five years, a visiting scientist at the IBM T.J. Watson Cener, NY 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. Currently he is an associate professor at the Department of Information Engineering at the Chinese University of Hong Kong. Dr. Wong is a senior member of the IEEE.