Staggered Push—A Linearly Scalable Architecture for Push-Based Parallel Video Servers

Jack Y. B. Lee, *Member, IEEE*

***Abstract*—With the rapid performance improvements in low-cost PCs, it becomes increasingly practical and cost-effective to implement large-scale video-on-demand (VoD) systems around parallel PC servers. This paper proposes a novel parallel video server architecture where video data are striped across an array of autonomous servers connected by an interconnection network. To coordinate data transmissions from multiple autonomous servers to a client station, a staggered push scheduling algorithm is proposed. A system model is constructed to quantify the performance of the architecture. Unlike most studies, this work does not assume the existence of a global clock among the servers and tackles two problems arising from server asynchrony: inconsistent schedule assignment and traffic overlapping. The former problem is solved by using an admission scheduler and the latter problem is solved by an over-rate transmission scheme. Analytical results prove a remarkable property of the staggered push architecture: as long as the network has sufficient capacity, the system can be scaled up linearly to an arbitrary number of servers. Design examples and numerical results are used to evaluate the proposed architecture under realistic assumptions and to compare it against other architecture.**

***Index Terms*—Parallel video server, performance analysis, scheduling algorithm, server push, server striping, staggered push, video-on-demand.**

I. Introduction

TRADITIONAL video-on-demand (VoD) systems commonly employ a high-performance server for the storage and delivery of video streams to multiple clients. This single-server approach is a natural extension of networked file systems and works well for small-scale systems with medium-quality videos. However, with the emergence of high definition video in terrestrial broadcasting, consumers will increasingly demand similar high quality video from VoD service providers. Coupled with the need to serve a large number of concurrent users, the capacity of the single-server approach is quickly becoming a severe limiting factor.

While server replication [1]–[3] and partition can be used to scale up the system capacity, the former approach would not be economical due to the large storage required for high-quality videos; and the latter approach is known to suffer from load-balancing problems [4], [5].

In this paper, we propose a parallel-server architecture for designing scalable VoD systems. Unlike replication, we use striping to achieve load sharing across multiple servers without increasing storage requirement. Furthermore, by striping using a small unit size, the system is insensitive to skewness in video retrievals. This architecture allows one to incrementally scale up the system capacity to more concurrent users by adding (rather than replacing) more servers and redistributing (rather than duplicating) video data among them.

The main contributions of this paper are as follows.

- We propose and analyze quantitatively a staggered push architecture for scheduling disk retrieval and network transmission in parallel video servers. We prove a remarkable property of the staggered push architecture—the system can be scaled up *linearly* to an arbitrary number of servers as long as the network has sufficient capacity.
- We discover that for loosely-coupled servers like PC or workstation clusters, server-clock asynchrony could lead to inconsistent schedule assignments among different servers. To tackle this problem, we propose the addition of an external admission scheduler to centralize admission control and perform schedule assignments.
- Apart from inconsistent schedule assignments, we discover that server-clock asynchrony could also lead to overlapping between data transmitted from different servers. This could induce network congestion, leading to video packets being dropped at the network switches and routers. Worst still, the client may not be able to cope with the aggregate data rate even if the network can successfully deliver the data. To tackle this problem, we propose an over-rate transmission scheme that can effectively prevent traffic overlapping.
- To evaluate the strengths and weaknesses of the proposed architecture, we use numerical results to compare and contrast staggered push with another architecture—concurrent push [6], using the same system parameters and assumptions.

The rest of the paper is organized as follows. Section II reviews some related works and compares them with this study. Section III presents an overview of the system architecture. Section IV studies the inconsistent schedule assignment problem and proposes an admission scheduler to tackle the problem. Section V studies the traffic overlapping problem and proposes an over-rate transmission scheme to tackle the problem. Section VI presents buffer management algorithms for server and client,

Manuscript received March 24, 1999; revised August 13, 2002. This work was supported in part by Research Grant CUHK6095/99E from the HKSAR Research Grant Council and the Area of Excellence in Information Technology, and a research grant from the HKSAR University Grants Council. The associate editor coordinating the review of this paper and approving it for publication was Prof. Chung-Yu Wu.

The author is with the Department of Information Engineering, Chinese University of Hong Kong, Shatin, N.T., Hong Kong (e-mail: jacklee@computer.org).

Digital Object Identifier 10.1109/TMM.2002.806533

and derives the respective buffer requirement. Section VII evaluates the system performance using numerical results and compares to the concurrent-push architecture. Section VIII discusses practicality and reliability issues. Finally Section IX draws a conclusion.

II. Related Works

Recently, a number of researchers have proposed and studied various architectures for implementing parallel video servers. An overview with architectural comparisons can be found in [7]. For most studies, fair and meaningful quantitative comparisons are not feasible due to the inherent differences in architecture, assumptions, and the lack of compatible performance models. Therefore we first discuss the relevant literatures in this section and compare them qualitatively with the approach proposed in this paper. A quantitative comparison with the concurrent push architecture [6] will be presented in Section VII.

First, the studies by Buddhikot *et al.* [8], Lougher *et al.* [9], Tewari *et al.* [10], [11], and Wu *et al.* [12] are based on architectures having two types of nodes, namely storage nodes and delivery nodes (called independent-proxy in [7]). In [9]–[12], the delivery nodes are independent hosts connected to all servers via a high-speed interconnect. The delivery nodes merge video data retrieved from the storage nodes and deliver them to the clients. In the study by Buddhikot *et al.* [8], the delivery node is a proprietary ATM switch (called APIC), and is responsible for delivering data blocks retrieved by the storage nodes in a synchronous manner to a client. Effectively, the APIC provides the hardware global clock needed to precisely synchronize the storage nodes.

This independent-proxy architecture is fundamentally different from the architecture proposed in this paper, where servers directly transmit video data to a client without passing through any intermediate node. Staggered push eliminates the extra hardware needed to run the intermediate delivery nodes as well as the extra high-speed interconnect linking up the storage nodes with the delivery nodes.[1] By incorporating the effect of server clock jitter and compensating accordingly, existing network hardwares such as FastEthernet and ATM can be used. Results (see Section VII) show that the staggered push architecture is robust to server clock jitter and performs well even if servers are loosely synchronized using conventional distributed clock synchronization algorithms.

The studies by Freedman *et al.* [13] and Lee *et al.* [14] are based on the client-pull service mode, where the server process video-block-level requests as they arrive. This model differs from the server-push service model employed in staggered push. A detail comparison between the two service models is beyond the scope of this paper. Briefly speaking, client-pull results in simpler server design and does not need server-clock synchronization. On the other hand, server-push allows better optimization of server efficiency as periodic schedulers can be used. Moreover, a full-fledged up-stream communication channel is not required, as there are no periodic requests traveling from a client back to the servers. Interested readers are referred to Rao *et al.* [15] for qualitative and simulation comparisons of the two service models in single-server multimedia systems.

Finally, the studies by Biersack *et al.* [16], [17], Bolosky *et al.* [18], Lee [6], and Reddy [19] are more closely related to the architecture proposed in this paper. In particular, the servers in these studies all send data directly to the clients (called proxy-at-client in [7]). Secondly, these studies all employ some forms of server-push service model.

This paper differs from the study by Biersack *et al.* [16], [17] in three ways. First, they proposed to stripe video across servers in fixed number of frames while we propose striping using fixed-size blocks. As compressed video frames vary widely in sizes, the former approach clearly has potential storage and load balancing problem.[2] By contrast, striping using fixed-size blocks avoids the frame-level processing issues and guarantees storage balance irrespective of the compression formats. Second, while they also suggest striping in fixed-size units in [17], their study did not consider variations in video-block-consumption times and simply assumed constant consumption time. This assumption is not valid for most compression formats as each fixed-size block can contain different number of frames (even partial frames) [6]. Third, their study did not consider scheduling issue at the servers and assumed a server transmits data at line speed to a client, which is clearly impractical due to the client's lower processing capability. This study has investigated the server-scheduling problem, revealed the inconsistent schedule assignment problem and traffic overlapping problem arising from server clock jitter, and proposed solutions to solve them.

The study by Bolosky *et al.* [18] differs from this paper in two ways. First, they focused on experimentation by means of an implementation using the PC platform, and did not deal with performance modeling in detail. Second, while their design also employs a centralized controller for admission control, they have not discovered the inconsistent schedule assignment problem arising from server-clock asynchrony. By contrast, we focus on modeling the system performance and on quantifying the effect of server-clock asynchrony. We reveal the inconsistent schedule assignment problem as well as the traffic overlapping problem and tackle them with an admission scheduler and over-rate transmission, respectively.

For the study by Reddy [19], while the author did point out that potential problems could occur in case the server clocks are not synchronized, no solution was suggested. Moreover, the proposed architecture is designed for a specific interconnection switch (Omega network). The study assumed that all server clocks are precisely synchronized and transmission via the switch can be exactly scheduled. By contrast, this paper assumes a loosely-coupled system and derives a more realistic performance model that incorporates server clock asynchrony, delay jitters, and variable video consumption rates.

[1]The independent proxy can also be implemented within the storage servers—called *proxy-at-server*. While this does not require additional hosts dedicated for the proxies, transmission and processing overheads are still incurred as the embedded proxy module will still have to receive video data from all other servers and then forward to the clients.

[2]While one can perform striping in units of group of pictures (GOP) for MPEG-compressed video, GOPs still varies slightly in size. Worst, GOP structure (i.e., the sequence of I, B, P frames) in MPEG videos can change dynamically (e.g., to adapt to scene changes or increase in motions). Hence, this approach still suffers from potential storage and load balancing problem.

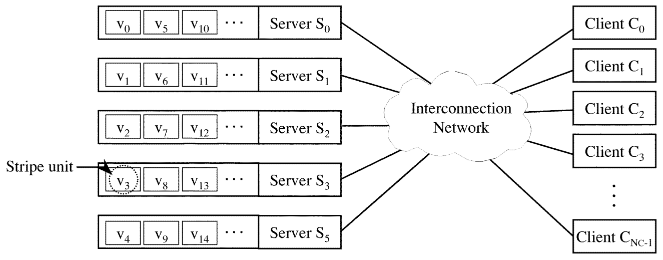

Fig. 1. Architecture of a (five-servers) parallel video server.

Finally, a related study by Lee [6] has proposed a scheduling algorithm called concurrent push for scheduling servers' retrieval and transmission. Briefly speaking, all servers transmit to a client concurrently and continuously under concurrent push. The transmission rate is reduced proportionally so that the aggregate data rate is equal to the average video bit-rate. The study proposed an Asynchronous Group Sweeping Scheme (AGSS) to reduce client buffer requirement and system response time. Nonetheless, the client buffer requirement still increases with system scale (i.e., number of servers) and the author proposed a Sub-schedule Striping Scheme (SSS) to maintain a constant buffer requirement, at the expense of higher processing overhead at the client.

This concurrent-push algorithm is designed to take advantage of the quality-of-service (QoS) control available in today's ATM networks. In particular, as a server sends data continuously to a client at a constant rate, one can easily integrate the constant-bit-rate (CBR) service available in today's ATM networks so that end-to-end QoS can be guaranteed. The tradeoff, however, is scalability—both server buffer requirement and client processing overhead increase with system scale. By contrast, the staggered push algorithm proposed in this paper is linearly scalable, i.e., the server requirement and client requirement remains the same irrespective of the system scale. While staggered push cannot take advantage of ATMs QoS control, we can still tackle network congestion problems by the over-rate transmission scheme proposed in Section V.

III. System Architecture

Fig. 1 shows the architecture of a parallel video server, comprising multiple autonomous servers connected by an interconnection network. We denote the number of servers in the system by N_S and the number of clients by N_C. Hence the client–server ratio, denoted by Λ, is N_C/N_S. Each server has separate CPU, memory, disk storage, and network interfaces. A server's storage spaces are divided into fixed-size stripe units of Q bytes each. Each video title is then striped into blocks of Q bytes and stored into the servers in a round-robin manner as shown in Fig. 1.

Striping with fixed-size blocks simplifies the process of striping video streams encoded using interframe compression algorithms (e.g., MPEG), where frame size varies considerably for different frame types. Since a stripe unit is significantly smaller than a video title (kilobytes versus megabytes), this enables fine-grain load sharing (as oppose to coarse-grain load sharing in data partitioning) among servers. Moreover, the loads are evenly distributed to all servers irrespective of skewness in video popularity [5].

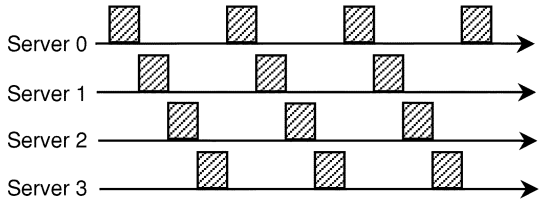

Fig. 2. Transmission scenario for the staggered push algorithm.

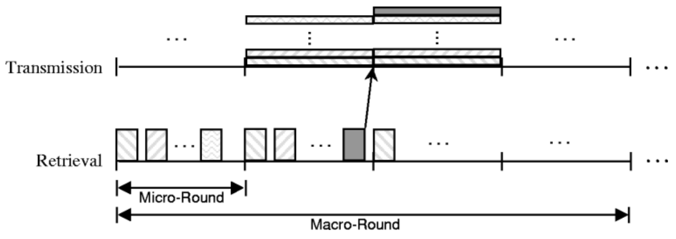

Fig. 3. Two-level scheduler for staggered push.

To schedule disk retrievals and network transmissions at the servers, we propose a *staggered push* algorithm where the servers transmit bursts of data to a client in a round-robin manner at the average video bit rate. Let R_V be the average video rate and assumed to be the same for all clients. Then the transmissions from the servers are staggered such that only one of the servers transmits to a receiver at any given time, depicted in Fig. 2. In this way, there will be at most $\Lambda = N_C/N_S$ video blocks being transmitted concurrently at a server. Note that while one can potentially reduce server buffer requirement by transmitting at a rate higher than R_V, the client in turn will have to be capable of receiving at such a high data rate. This is less practical as client network connection usually has lower bandwidth and the client device (e.g., set-top box) will likely have limited processing capability.

To support staggered push, the server scheduler is divided into two scheduling levels: *micro-round* and *macro-round* as shown in Fig. 3. Video blocks retrieved in a micro-round will be transmitted in the next micro-round. Let T_F be the average time needed to completely transmit a video block of Q bytes. Since a video block is transmitted at a rate equal to the video data rate R_V, we can obtain T_F from

$$T_F = Q/R_V. \tag{1}$$

In an N_S-servers system, each macro-round consists of N_S micro-rounds, and each micro-round transfers Λ video blocks. Hence, the disk will transfer up to $N_S\Lambda = N_C$ video blocks in one macro-round, with one block transmitted for each video stream.

IV. Schedule Assignment

Unexpectedly, the two-level scheduling scheme may result in inconsistent schedules among different servers if admission is performed independently at each server. Specifically, as servers are loosely coupled, the internal clock of each server in the system will not be precisely synchronized. We define *clock jitter*

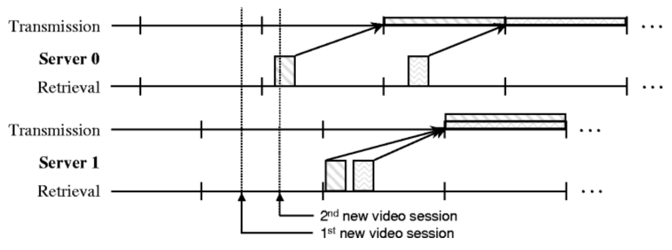

Fig. 4. Inconsistent schedule assignment arising from server clock jitter.

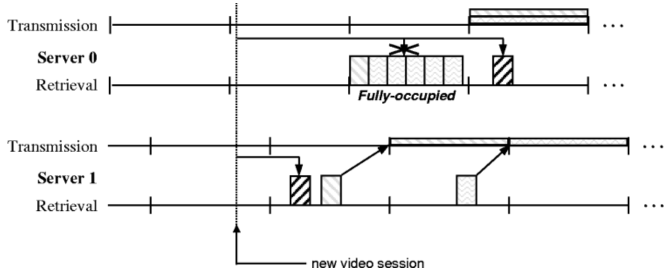

Fig. 5. Micro-round overflow due to inconsistent schedule assignment.

as the difference between the internal real-time clocks of two servers. Many algorithms for controlling clock jitter between distributed computers have been proposed [20]–[22] and hence will not be pursued further here. We assume that the maximum clock jitter between any two servers in the system is bounded and is denoted by τ.

With the presence of clock jitter, one server could assign two new video sessions to start with the same micro-round while another server could assign them to two different micro-rounds as shown in Fig. 4. This can occur because each server assigns new sessions to micro-rounds according to its own internal clock, which differs from other servers due to clock jitter. As a single micro-round can serve only up to Λ video sessions, eventually one server could experience micro-round overflow although another server can admit the new video session (Fig. 5). While one can delay the new video session at the overflowed server until the next available micro-round, the transmission schedule will be delayed significantly and will result in severe traffic overlapping with transmissions from another server (see Section V).

To solve this inconsistent schedule assignment problem, we propose adding an external admission scheduler between the servers and the clients to centralize schedule assignment. To initiate a new video session, a client will first send a request to the admission scheduler. Using the same clock-synchronization protocol, the admission scheduler maintains the same clock jitter bound with the servers. As new sessions are assigned solely according to the admission scheduler's clock, the scenario depicted in Figs. 4 and 5 will not occur. However, to ensure that the assigned micro-round has not started in any of the servers due to clock jitter, the admission scheduler must add an extra delay to the assignment.

Theorem 1: If the admission scheduler delays the start of a new video session by

$$\Delta = \left\lceil \frac{\tau}{T_F} \right\rceil \tag{2}$$

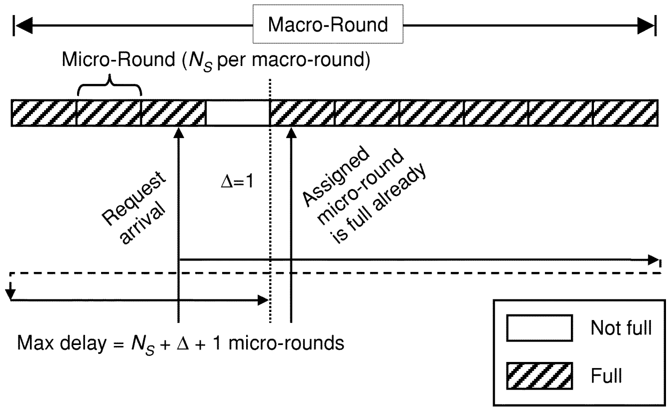
Macro-Round
Micro-Round (N_S per macro-round)
Request arrival
Δ=1
Assigned micro-round is full already
Max delay = N_S + Δ + 1 micro-rounds
Not full
Full

Fig. 6. Worst-case delay in the admission process.

micro-rounds, then it guarantees that the assigned micro-round has not started in any of the N_S servers.

Proof: See Appendix A. ■

For example, let t_A be the local time the new request arrives at the admission scheduler. Then the admission scheduler will attempt to admit the request to micro-round

$$n_A = \left\lfloor \frac{t_A}{T_F} \right\rfloor + 1 + \Delta. \tag{3}$$

Note that we need to add one to n_A because a new request cannot join the current micro-round (it has started already). If the assigned micro-round is full, the admission scheduler will sequentially check the subsequent micro-rounds until an available micro-round is found. In the worst case shown in Fig. 6, the transmission of the first video block will be delayed for $(N_S + \Delta + 1)$ micro-rounds:

$$D_S = T_F \left(N_S + \left\lceil \frac{\tau}{T_F} \right\rceil + 1 \right). \tag{4}$$

To better evaluate the delay incurred, we can derive the average scheduling delay under a given server load. Assume that there are n $(0 \le n \le \Lambda N_S)$ active video sessions, then it can be shown that (see Appendix B) the average scheduling delay is given by

$$D_S = \frac{Q}{R_V} \sum_{j=1}^{N_S-1} \left(\sum_{k=1}^{N_S} \frac{k(N_S-j)j!(N_S-k-1)!}{N_S!(j-k)!} + \left\lceil \frac{\tau}{T_F} \right\rceil + 1 \right) \left(\frac{N_{full}(n, j)}{N(n, N_S, \Lambda)} \right) \tag{5}$$

where

$$N(n, N_S, \Lambda) = \sum_{j=0}^{N_S} (-1)^j \binom{N_S}{j} \binom{N_S + n - j(\Lambda+1) - 1}{N_S - 1} \tag{6}$$

and

$$N_{full}(n, m) = \binom{N_S}{m} N(n - m\Lambda, N_S - m, \Lambda - 1). \tag{7}$$

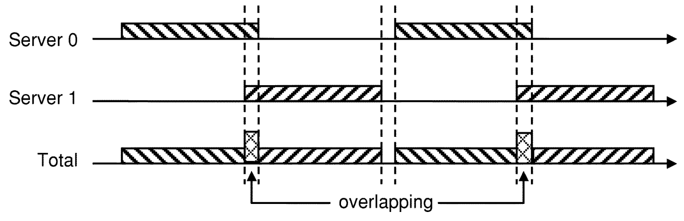

Fig. 7. Traffic overlapping due to server clock jitter.

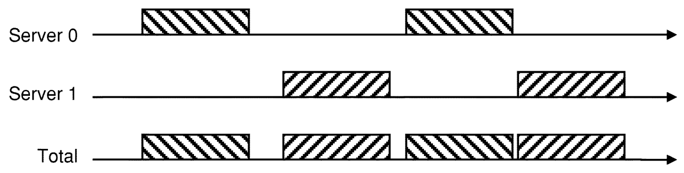

Fig. 8. Preventing traffic overlap by over-rate transmission.

V. Traffic Overlapping

If server clock jitter is larger than zero then transmissions from two or more servers destined to the same client will overlap and multiply the transmission rate in the overlapping interval (Fig. 7). This could cause congestion at the network and the client, resulting in packet being dropped.

To avoid traffic overlapping, we can sacrifice some server and network bandwidth, and transmit video data at a rate higher than R_V, say R_{ORT} (Fig. 8). We call this scheme over-rate transmission (ORT) for obvious reason. The transmission window will then be reduced to a time interval of

$$T_w = \frac{Q}{R_{ORT}}. \tag{8}$$

We can guarantee that there will be no transmission overlapping by ensuring that

$$T_w + \tau < T_F. \tag{9}$$

Rearranging, we can then obtain the minimum transmission rate needed to avoid traffic overlapping:

$$R_{ORT} > \frac{QR_V}{Q - R_V\tau}. \tag{10}$$

Since the transmission rate must be positive and less than infinity, we have the condition that

$$\tau < \frac{Q}{R_V} = T_F. \tag{11}$$

In other words, the server clock jitter must be smaller than a micro-round. Note that under this condition, traffic overlapping involves at most two servers and the data rate is doubled to $2R_V$ in the overlapping region. As R_{ORT} in (10) can become very large when the denominator becomes small, the useful operating range for over-rate transmission is actually limited by

$$R_{ORT} < 2R_V. \tag{12}$$

Substituting (10) into (12) and rearrange we can then determine the maximum clock jitter for which ORT is applicable:

$$\tau < \frac{Q}{2R_V} = 0.5T_F. \tag{13}$$

Therefore, ORT can prevent traffic overlapping if clock jitter is less than half of a micro-round. With ORT, the maximum network bandwidth needed at each server will be increased to

$$C_{ORT} = \Lambda R_{ORT} = \frac{\Lambda Q R_V}{Q - R_V\tau}. \tag{14}$$

VI. Buffer Management

In this section, we present buffer management algorithms employed at the server and client, and derive the respective buffer requirements. For simplicity, we ignore network delay and delay jitter. However, the effect of network delay and delay jitter can be incorporated in the same way as clock jitter and the same derivations are still valid.

A. *Server Buffer Requirement*

There are N_S micro-rounds in a macro-round, therefore the duration of a macro-round, denoted by T_R, is given by

$$T_R = \frac{N_S Q}{R_V}. \tag{15}$$

As buffers are released after each micro-round, this scheduler requires only $2\Lambda Q$ buffers for each server, regardless of the number of servers and clients in the system. Therefore, existing servers do not need any upgrade when one scale up a system by adding more servers.

B. *Client Buffer Requirement*

Many studies on VoD system assumed that video data are consumed periodically by the video decoder. However, our experience on programming some off-the-shelf hardware and software video decoders reveals that the decoder consumes fixed-size data blocks only quasiperiodically [14]. Given the average video data rate, R_V, and block size, Q, the average time for a video decoder to consume a single block is

$$T_{avg} = \frac{Q}{R_V}. \tag{16}$$

To quantify the randomness of video block consumption time, we employ the consumption model proposed in [6], reproduced below for sake of completeness.

Definition 1: Let T_i be the time the video decoder starts decoding the ith video block, then the decoding-time deviation of video block i is defined as

$$T_{DV}(i) = T_i - iT_{avg} - T_0 \tag{17}$$

and decoding is late if $T_{DV}(i) > 0$ and early if $T_{DV}(i) < 0$. The maximum lag in decoding, denoted by T_L, and the maximum advance in decoding, denoted by T_E, are defined as follows:

$$T_L = \max\{T_{DV}(i)\,|\,\forall i \geq 0\} \tag{18}$$

$$T_E = \min\{T_{DV}(i)\,|\,\forall i \geq 0\}. \tag{19}$$

The bounds T_L and T_E are implementation dependent and can be obtained empirically. Knowing these two bounds, the playback instant for video block i, denoted by $p(i)$, is then bounded by

$$\max\{(T_0+iT_{avg}+T_E), 0\} \leq p(i) \leq (T_0+iT_{avg}+T_L). \tag{20}$$

Buffers are used at the client to absorb these variations to prevent buffer underflow (which leads to playback hiccups) and buffer overflow (which leads to packet dropping). Let $L_C = (Y + Z)$ be the number of buffers (each Q bytes) available at the client, organized as a circular buffer. The client prefills the first Y buffers before starting playback to prevent buffer underflow, and reserves the last Z buffers for incoming data to prevent buffer overflow.

We first determine the lower bound for Y. Let t_0 be the time (with respect to the admission scheduler's clock) when the first block of a video session begins transmission. Let d_i be the clock jitter between the admission scheduler and server i. Without loss of generality, we can assume that the video title is striped with block zero at server zero. Then the time for block i to be completely received by the client, denoted by $f(i)$, is then bounded by

$$\begin{aligned}&\left((i+1)T_F + t_0 + f^- + d_{\mathrm{mod}(i,\,N_S)}\right)\\ &\qquad \le f(i) \le \left((i+1)T_F + t_0 + f^+ + d_{\mathrm{mod}(i,\,N_S)}\right)\end{aligned} \tag{21}$$

where f^+ and f^- are used to model the maximum transmission time deviation due to randomness in the system, including transmission rate deviation, CPU scheduling, bus contention, etc.

Since the client begins video playback after filling the first Y buffers, the playback time for video block 0 is simply equal to $f(Y-1)$. Setting $T_0 = f(Y-1)$ in (20) then the playback time for video block i is bounded by

$$(f(Y-1) + iT_{avg} + T_E) \le p(i) \le (f(Y-1) + iT_{avg} + T_L). \tag{22}$$

To guarantee video playback continuity, we must ensure that all video blocks arrive before their respective playback deadlines. Therefore, we need to ensure that for all video blocks, the latest arrival time must be smaller than the earliest playback time:

$$\max\{f(i)\} < \min\{p(i)\} \qquad \forall\, i \ge 0. \tag{23}$$

Using the bounds from (21) and (22), we can rewrite (23) as

$$(i+1)T_F + t_0 + f^+ + d_{\mathrm{mod}(i,\,N_S)} < iT_{avg} + f(Y-1) + T_E \tag{24}$$

$$\begin{aligned}&(i+1)T_F + t_0 + f^+ + d_{\mathrm{mod}(i,\,N_S)}\\ &\qquad < iT_{avg} + YT_F + t_0 + f^- + d_{\mathrm{mod}(Y-1,\,N_S)} + T_E.\end{aligned} \tag{25}$$

From (1) and (16), we know that $T_{avg} = T_F$, rearranging and solving for Y, we then obtain

$$Y > 1 + \left(\frac{f^+ - f^- - T_E + \left(d_{\mathrm{mod}(i,\,N_S)} - d_{\mathrm{mod}(Y-1,\,N_S)}\right)}{T_F}\right). \tag{26}$$

Since $\max\{|d_i - d_j|\,|\forall\, i,\, j\} = \tau$, the worst case is

$$Y > 1 + \left(\frac{f^+ - f^- - T_E + \tau}{T_F}\right) \tag{27}$$

which is the number of buffers that must be prefilled before beginning video playback.

Similarly, to guarantee that the client buffer will not be overflowed by incoming video data, we need to ensure that the ith video block starts playback before the $(i + L_C - 2)$th video block is completely received. This is because the client buffers are organized as a circular buffer. Therefore, we need to ensure that

$$\min\{f(i+L_C-2)\} > \max\{p(i)\} \qquad \forall\, i \ge (L_C - Z). \tag{28}$$

Again using the bounds from (21) and (22), we can rewrite (28) as

$$\begin{aligned}&(i + L_C - 1)T_F + t_0 + f^- + d_{\mathrm{mod}(i+L_C-2,\,N_S)}\\ &\qquad > iT_{avg} + f(L_C - Z - 1) + T_L\end{aligned} \tag{29}$$

or

$$\begin{aligned}&(i + L_C - 1)T_F + t_0 + f^- + d_{\mathrm{mod}(i+L_C-2,\,N_S)}\\ &\qquad > iT_{avg} + (L_C - Z)T_F + t_0 + f^+\\ &\qquad\quad + d_{\mathrm{mod}(L_C-Z-1,\,N_S)} + T_L.\end{aligned} \tag{30}$$

Similarly, rearrange and solve for Z we obtain (31), as shown at the bottom of the page. Again noting that $\max\{|d_i - d_j|\,|\forall\, i,\, j\} = \tau$, the worst case is

$$Z > 1 + \left(\frac{f^+ - f^- + T_L + \tau}{T_F}\right) \tag{32}$$

which is the number of empty buffers needed to avoid client buffer overflow.

C. System Response Time

Another key performance metric of a VoD service is system response time, defined as the time from initiating a new request to the time video playback starts. Ignoring system administration and network delay issues, system response time is consisted of two components: scheduling delay and prefill delay. Scheduling delay is the delay incurred at the admission scheduler plus the delay incurred at the server scheduler, as derived in Section IV. For prefill delay, we note that the client prefills the first Y video blocks before starting video playback. Hence, the average prefill delay can be obtained from

$$D_P = \frac{YQ}{R_V} \tag{33}$$

and the system response time is simply the sum $D_S + D_P$.

$$Z > 1 + \left(\frac{f^+ - f^- + T_L + \left(d_{\mathrm{mod}(L_C-Z-1,\,N_S)} - d_{\mathrm{mod}(i+L_C-2,\,N_S)}\right)}{T_F}\right) \tag{31}$$

TABLE I
SYSTEM PARAMETERS USED IN PERFORMANCE EVALUATION

System Parameters	Symbol	Value
Video block size	Q	65536 Bytes
Video data rate	R_V	150KB/s
Maximum early in decoding time	T_E	-130ms
Maximum late in decoding time	T_L	160ms
Client-Server ratio	Λ	10
Transmission time deviation	f^-, f^+	0ms
Server clock jitter	τ	100ms

VII. Performance Evaluation

In this section, we evaluate the performance of the proposed architecture using numerical results. All results are computed using the derivations in Sections IV–VI with the system parameters listed in Table I. The parameters T_E, T_L are determined experimentally by collecting video block consumption times of a hardware MPEG-1 decoder (Sigma Designs RealMagic).

A. Design Example

To illustrate performance and resource requirements of the architecture, we first consider a design example in this section. We assume that there are eight servers in the system, with a client–server ratio of 50[3] (i.e., up to 400 concurrent streams). Using the parameters in Table I, the server buffer requirement is calculated to be 6.25 MB. Compared with the amount of memory in today's PC, this buffer requirement is relatively small. Moreover, as conventional PCs can be expanded to 256 MB or more memory, in theory a client–server ratio of over 2000 can be supported. Hence server buffer requirement will not become a limiting factor to the system's scalability.

Using the same parameters, the client buffer requirement is calculated to be 256 KB. This translates into an average prefill delay of 1.41 s. To determine the system response time, we assume that the system is at 90% utilization. Then the corresponding scheduling delay will be 0.735 s. Together with prefill delay, the average system response time becomes 2.146 s, well within acceptable limits. We perform more detailed sensitivity analysis with respect to key system parameters in the following sections.

B. Server Buffer Requirement

Fig. 9 plots server buffer requirement versus system scale (i.e., number of servers) for both concurrent push and staggered push. This graph clearly shows the remarkable property of staggered push—constant server buffer requirement irrespective of system scale. By contrast, server buffer requirement increases with system scale under concurrent push, even with AGSS and SSS. When concurrent push is scaled up to 12 servers, server buffer requirement increases to 40.6 MB compared to just 6.25 MB under staggered push. Hence the ultimate scalability of the concurrent push architecture will be limited by server buffer, while the proposed staggered push architecture can be scaled up without any upgrade to the existing servers.

[3]This particular client–server ratio is determined from past implementation experiences using PentiumPro-200 Mhz class machines.

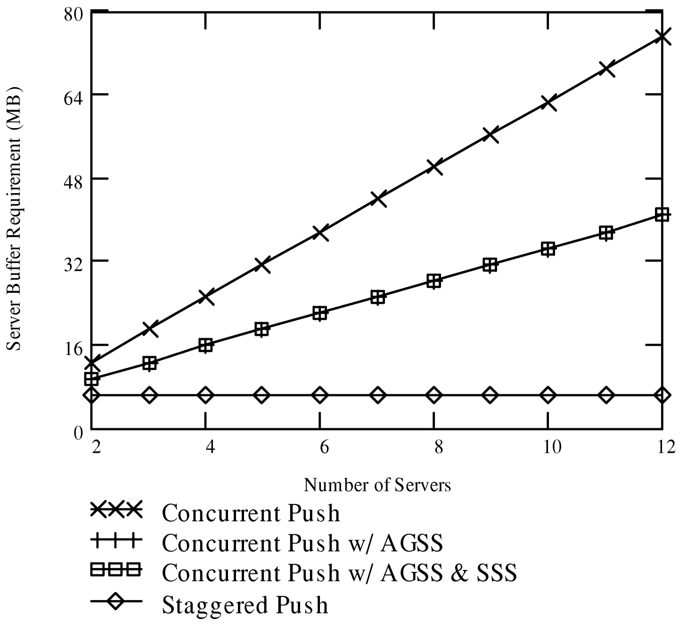

Fig. 9. Server buffer requirement versus system scale.

C. Client Buffer Requirement

Fig. 10 plots client buffer requirement versus system scale for both concurrent push and staggered push. We observe that concurrent push is not scalable without SSS, while staggered push has a constant client buffer requirement that will not limit scalability. Note that although client buffer requirement in concurrent push can be controlled to a constant by SSS [6], the system scalability is still limited as client processing overhead due to SSS increases with system scale. It is particularly important to maintain a constant client buffer requirement in practice as it would be very expensive (if not impossible) to upgrade every existing client devices (e.g., set-top box) whenever the system is scaled up.

In Fig. 11, we analyze the sensitivity of client buffer requirement to server clock jitter. As the results indicate, the buffer requirement is relatively insensitive to clock jitter, even if the jitter is increased to one second. Hence one can safely employ the existing software-based, distributed clock-synchronization protocols in staggered push.

D. System Response Time

Fig. 12 plots the system response time versus system scale. While the worst-case system response time increases linearly with more servers, the average system response time remains

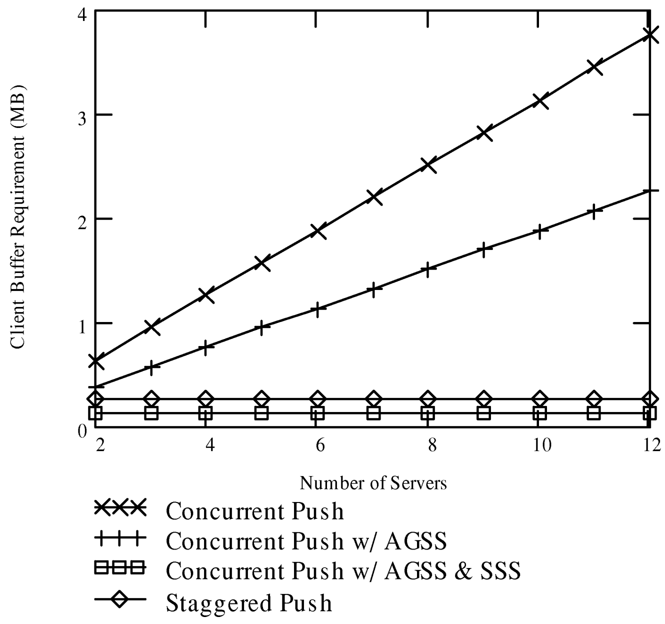

Fig. 10. Client buffer requirement versus system scale.

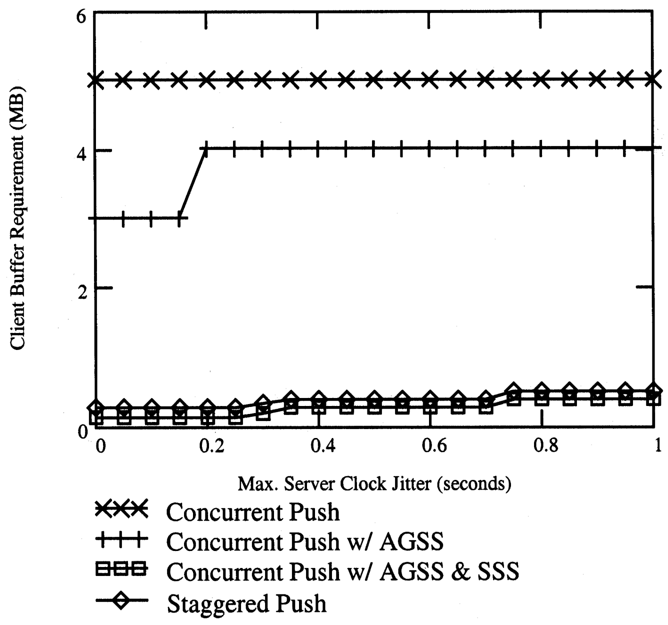

Fig. 11. Client buffer requirement versus server clock jitter.

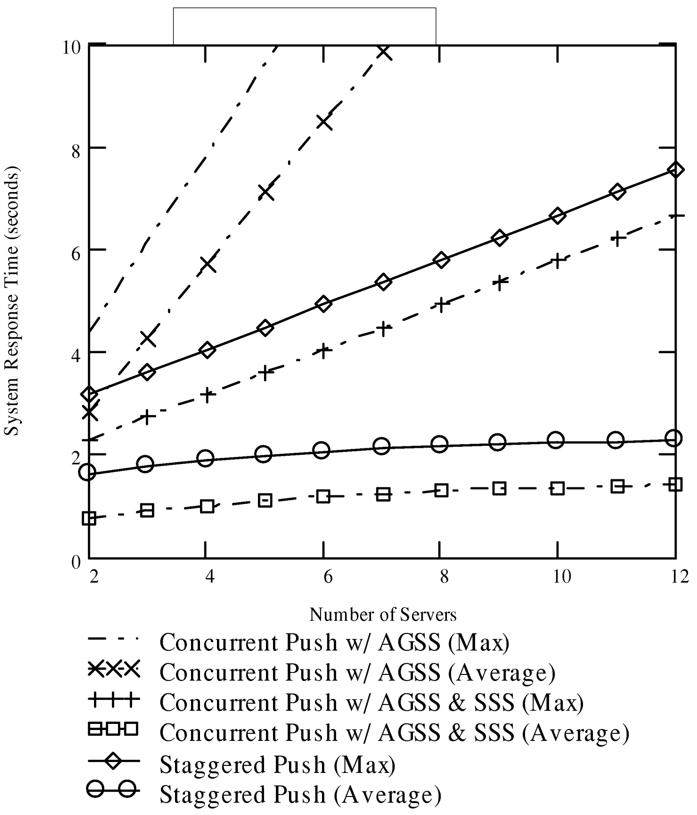

Fig. 12. Average system response time versus system scale at (90% utilization).

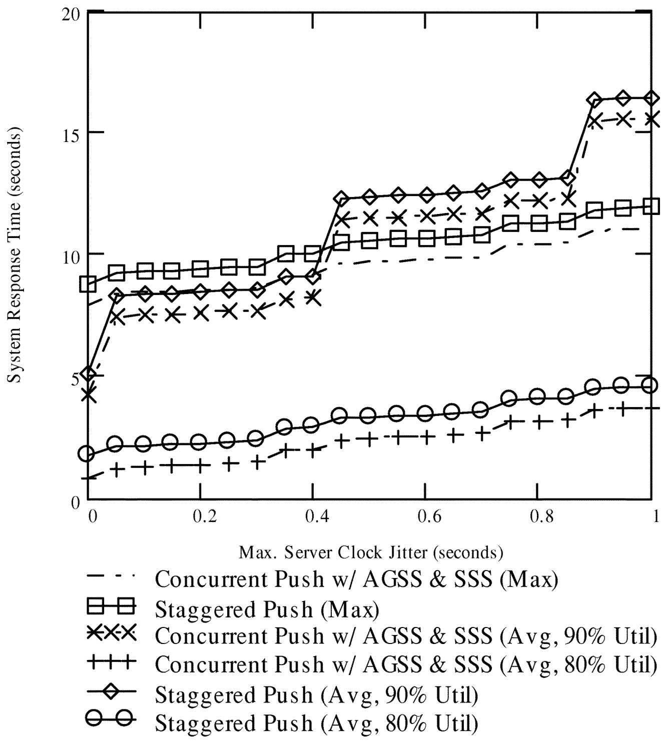

Fig. 13. System response time versus server clock jitter.

low (~2 s) for a utilization of 90%. This suggests that we can maintain a low system response time simply by limiting the system to, say, 90% utilization by means of admission control.

In Fig. 13, we study the sensitivity of system response time to server clock jitter. As expected, the system response time increases for larger clock jitter values (cf. Theorem 1). However given that server clock jitter can readily be controlled to within 100 ms [20], the average system response time is still only 0.735 s for an eight-servers system at 90% utilization.

E. Server Bandwidth Overhead

Fig. 14 plots the ORT transmission rate versus server clock jitter for block sizes of $Q = 64$ KB, 128 KB, and 256 KB. As clock jitter can be readily controlled to within 100 ms by distributed software algorithms, the results show that over-rate transmission is applicable to all three cases. For example, with $Q = 64$ KB, ORT will transmit at 1.556 Mbps instead of the video bit rate at 1.2 Mbps, incurring a bandwidth overhead of

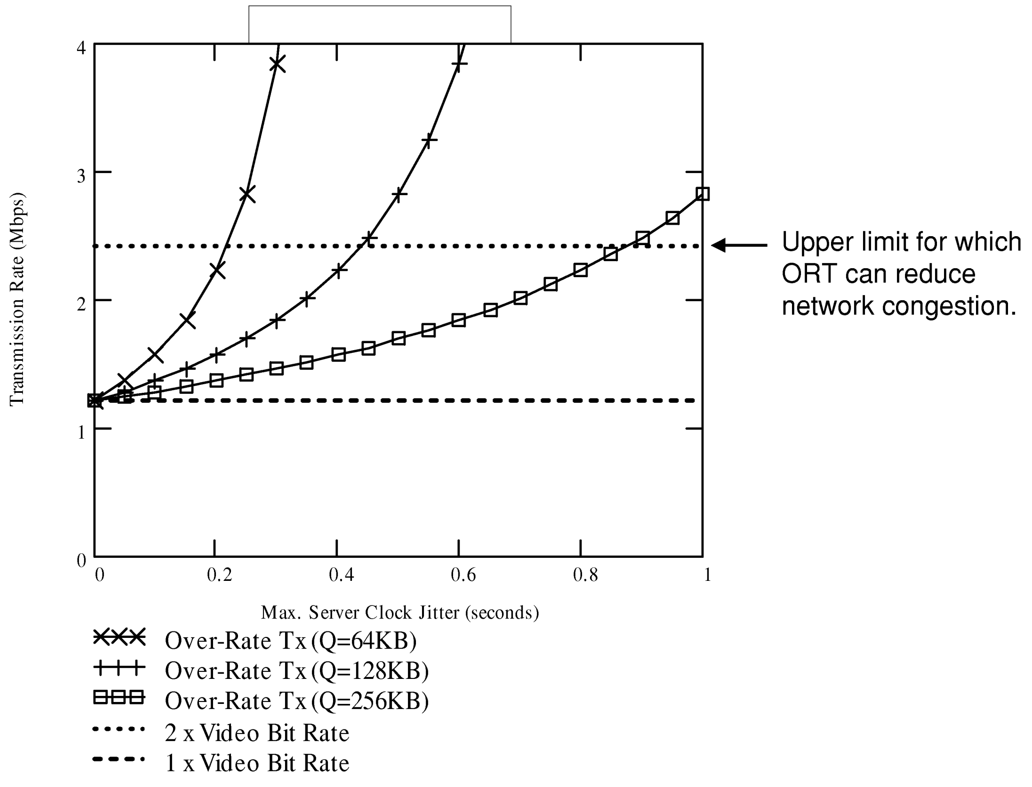

Fig. 14. Transmission rate versus server clock jitter.

29.7%. Increasing the block size to 256 KB reduces the ORT transmission rate to 1.273 Mbps, or a bandwidth overhead of only 6%. Thus the system designer can adjust the block size to balance between bandwidth cost and memory cost. In any case, compared to uncontrolled traffic overlapping which results in doubled transmission rate at 2.4Mbps, bandwidth under ORT is clearly substantially lower.

VIII. Discussions

A. Practicalities

As the results in the previous section show, the proposed staggered push architecture can be scaled up to any number of servers, provided that the network has sufficient capacity. Compare with the concurrent-push architecture, staggered push achieves linear scalability at the expense of bursty network traffic (and slightly larger delay and client buffer requirement). In particular, if we consider the network traffic between a server and a client, it is easy to see that the traffic will be in the form of bursts with an average interburst interval of $(N_S - 1)T_F$ s. By contrast, servers in concurrent push transmit to a client continuously at a constant rate, allowing easy integration with QoS offered by existing ATM networks. Staggered push will not be able to make use of QoS available in today's ATM networks.

In practice, if the VoD system is deployed in dedicated networks with *a priori* bandwidth planning, then staggered push can still be used effectively. This is because the over-rate transmission scheme already guarantees that network congestion due to traffic overlapping will not occur, and the aggregate traffic going from the servers to a client will be close to constant bit-rate, with small gaps in between (due to over-rate transmission). On the other hand, Bolosky *et al.* [18] have suggested that future ATM networks may have support for such many-to-one traffic model that can provide QoS similar to the one available to current constant-bit-rate service.

B. Reliability

Another issue in parallel video server is reliability. Specifically, as videos are striped across servers without data redundancy, any single server failure will cripple the entire system. There are a number of ways to tackle this reliability problem. One approach, as proposed by Bolosky *et al.* [18], improves reliability by data mirroring. Their system replicates all video stripe units and distributes them to the servers using declustering so that additional loads after a server failure are evenly shared by the remaining servers. The obvious tradeoff is doubled storage requirement. A subtler tradeoff is the need for declustering. As no known algorithm can automatically produce a declustering scheme (i.e., where to place each replicated units) for an arbitrary number of servers, this mirroring approach would require more capacity planning when being scaled up.

Another approach is by means of parity units, as proposed by Lee *et al.* [24]. Their system introduces redundant units computed from video data units into the servers, and uses a special video transfer protocol to detect server failure. The client, armed with the redundant units and the survived data units, can then compute the lost video units in real-time. In the simplest form, the redundant units are simply parity units, computed from exclusive-or between the video data units of the same stripe. This parity-based striping scheme can protect single-server failure and their protocol can maintain continuous video playback despite server failure by means of additional buffering at the client. While their architecture differs from staggered push (pull-based versus push-based), similar redundant striping scheme can also be introduced to staggered push to achieve fault tolerance. The author is currently investigating the supportive system modules (e.g., fault-detection protocol, recovery protocol, transmission scheduling, etc.) that are needed to support fault tolerance in staggered push.

IX. Conclusion

In this paper, we propose and analyze a parallel video server architecture for implementing linearly scalable video-on-demand systems. The proposed architecture employs fixed-size block striping for data storage, and a staggered push scheduling algorithm for co-ordinating transmissions among multiple autonomous servers. We incorporate the effect of server clock jitter and reveal the inconsistent schedule assignment problem and the traffic overlapping problem. We tackle the former problem by an external admission scheduler and the latter problem by an over-rate transmission scheme. Our results show that the over-rate transmission scheme can effectively prevent traffic overlapping with a small bandwidth overhead under clock jitter bounds achievable by existing software-based synchronization algorithms. Moreover, we show that the server buffer requirement, the client buffer requirement, and the server bandwidth requirement are all independent of the number of servers in the system. The average system response time, though increases slightly with more servers, remains acceptable if we limit the system to less than full utilization. These results demonstrate that the proposed architecture can be scaled up to large number of users without costly upgrade to the existing servers and clients.

Appendix

A. Proof of Theorem 1

Let t_i be the local time a new request arrives at server i ($0 \le i < N_S$), t_A be the local time the new request arrives at the admission scheduler, and Δ be the extra scheduling delay (in number of micro-rounds). Then the admission scheduler will attempt to admit the request to micro-round n_A as given in (3). For server i, the new request arrives during micro-round $n_i = \lfloor t_i/T_F \rfloor$. Hence the problem is to find Δ such that $n_A > n_i$ for $0 \le i < N_S$, i.e., the assigned micro-round has not been started in any of the servers. Using this condition, we can then obtain the following inequality:

$$n_A > n_i. \tag{34}$$

Expanding gives

$$\left\lfloor \frac{t_A}{T_F} \right\rfloor + 1 + \Delta > \left\lfloor \frac{t_i}{T_F} \right\rfloor. \tag{35}$$

Rearranging gives

$$\Delta > \left\lfloor \frac{t_i}{T_F} \right\rfloor - \left\lfloor \frac{t_A}{T_F} \right\rfloor - 1. \tag{36}$$

Applying the inequality $\lfloor x \rfloor - \lfloor y \rfloor \le \lceil x - y \rceil$: $x, y \ge 0$, to the right-hand side of (36), we then obtain

$$\left\lfloor \frac{t_i}{T_F} \right\rfloor - \left\lfloor \frac{t_A}{T_F} \right\rfloor - 1 \le \left\lceil \frac{t_i}{T_F} - \frac{t_A}{T_F} \right\rceil - 1. \tag{37}$$

Since clock jitter is bounded: $|t_i - t_A| \le \tau$, for $0 \le i < N_S$, we can rewrite (37) in terms of τ:

$$\left\lceil \frac{t_i}{T_F} - \frac{t_A}{T_F} \right\rceil - 1 \le \left\lceil \frac{\tau}{T_F} \right\rceil - 1. \tag{38}$$

Hence, if

$$\Delta > \left\lceil \frac{\tau}{T_F} \right\rceil - 1 \tag{39}$$

or at least

$$\Delta = \left\lceil \frac{\tau}{T_F} \right\rceil \tag{40}$$

then the assigned micro-round is guaranteed to have not started in any of the servers. ■

B. Derivation of the Average Scheduling Delay

Assume that video sessions start independently and with equal likelihood at any time. Then a video session can be assigned to any one of the N_S micro-rounds with equal probability. Assume that there are n active video sessions, then the number of ways to distribute these n video sessions among N_S groups is a variant of the urn-occupancy distribution problem and is given by [25] as:

$$N(n, N_S, \Lambda) = \sum_{j=0}^{N_S} (-1)^j \binom{N_S}{j} \binom{N_S + n - j(\Lambda+1) - 1}{N_S - 1}. \tag{41}$$

To obtain the probability of having m fully-occupied micro-rounds, we first notice that there are $\binom{N_S}{m}$ possible combinations of having m fully-occupied micro-rounds. Given that, the number of ways to distribute $(n - m\Lambda)$ video sessions among $(N_S - m)$ micro-rounds with none of those micro-rounds fully-occupied can be obtained from (41) as $N(n - m\Lambda, N_S - m, \Lambda - 1)$. Hence the total number of ways for exactly m of the micro-rounds fully occupied is given by

$$N_{full}(n, m) = \binom{N_S}{m} N(n - m\Lambda, N_S - m, \Lambda - 1). \tag{42}$$

Hence, the probability of having m fully-occupied micro-rounds given n active video sessions can be obtained from

$$P_{full}(n, m) = \frac{N_{full}(n, m)}{N(n, N_S, \Lambda)}. \tag{43}$$

Assume that m out of N_S micro-rounds are fully occupied, then, the probability for the assigned micro-round to be available (not fully occupied) is given by

$$V_0 = \frac{N_S - m}{N_S}. \tag{44}$$

Hence $P_0 = (1 - V_0)$ will be the probability of the assigned micro-round being fully occupied. Now provided that the assigned micro-round is fully occupied, the probability that the next micro-round is available is

$$V_1 = \Pr\{\text{next round available} \mid P_0\} = \frac{N_S - m}{N_S - 1}. \tag{45}$$

This is also the probability for a client to wait one additional micro-round provided the assigned micro-round is already fully occupied. It can be shown that the probability for a client to wait k additional micro-rounds provided that the first k assigned micro-rounds are all fully occupied is

$$V_k = \Pr\{(k+1)\text{th round available} \mid P_k\} = \frac{N_S - m}{N_S - k} \quad 1 \le k \le m. \tag{46}$$

We already know P_0, and it can be shown that the probability for the first k micro-rounds all fully occupied is given by

$$P_k = \prod_{i=0}^{k-1} \left(\frac{m-i}{N_S - i} \right) = \frac{m!(N_S - k)!}{N_S!(m-k)!}, \qquad 1 \le k \le m. \tag{47}$$

Hence, we can solve for the probability of a client having to wait k additional micro-rounds from

$$\begin{aligned} W_k &= \Pr\{(k+1)\text{th round free} \mid P_k\} P_k \\ &= \frac{(N_S - m)m!(N_S - k - 1)!}{N_S!(m-k)!}, \qquad 1 \le k \le m. \end{aligned} \tag{48}$$

Therefore given m—the number of micro-rounds that are fully-occupied, the average number of micro-rounds a client has to wait can be obtained from

$$W_{avg}(m) = \sum_{k=1}^{N_S} k W_k + \left(\left\lceil \frac{\tau}{T_F} \right\rceil + 1 \right) \tag{49}$$

where the second term accounts for the additional delay as described in Theorem 1. Similarly, given n—the number of active video sessions, the average number of micro-rounds a client has to wait can be obtained from

$$M_{avg}(n) = \sum_{j=1}^{N_S - 1} W_{avg}(j) P_{full}(n, j). \tag{50}$$

And the corresponding average scheduling delay given a system utilization of n is

$$D_S = \frac{M_{avg}(n) Q}{R_V}. \tag{51}$$

Substituting (43), (48)–(50) into (51) gives the desired result. ■

ACKNOWLEDGMENT

The author would like to express his gratitude to the anonymous reviewers for their insightful comments and suggestions in improving this paper.

REFERENCES

[1] N. Venkatasubramanian and S. Ramanthan, "Load management in distributed video servers," in *Proc. 17th Int. Conf. Distributed Computing Systems*, Baltimore, MD, May 1997, pp. 528–535.

[2] C. C. Bisdikian and B. V. Patel, "Issues on movie allocation in distributed video-on-demand systems," in *Proc. ICC'95*, Seattle, WA, June 1995, pp. 250–255.

[3] S. A. Barnett and G. J. Anido, "A cost comparison of distributed and centralized approaches to video-on-demand," *IEEE J. Select. Areas Commun.*, vol. 14, no. 5, pp. 1173–1183, 1996.

[4] T. D. C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," *ACM Multimedia Syst.*, 1994.

[5] C. Griwodz, M. Bar, and L. C. Wolf, "Long-term movie popularity models in video-on-demand systems or the life of an on-demand movie," in *Proc. Multimedia'97*, pp. 349–357.

[6] Y. B. Lee, "Concurrent push—A scheduling algorithm for push-based parallel video servers," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, no. 3, Apr. 1999.

[7] ——, "Parallel video servers—A tutorial," *IEEE Multimedia*, vol. 5, no. 2, pp. 20–28, June 1998.

[8] M. M. Buddhikot and G. M. Parulkar, "Efficient data layout, scheduling and playout control in MARS," in *Proc. NOSSDAV'95*, 1995.

[9] P. Lougher, D. Pegler, and D. Shepherd, "Scalable storage servers for digital audio and video," in *Proc. IEE Int. Conf. Storage and Recording Systems 1994*, 1994, pp. 140–143.

[10] R. Tewari, D. M. Dias, R. Mukherjee, and H. M. Vin, "Real-time issues for a clustered multimedia servers,", IBM Res. Rep. RC20020, 1995.

[11] R. Tewari, R. Mukherjee, D. M. Dias, and H. M. Vin, "Design and performance tradeoffs in clustered video servers," in *Proc. 3rd IEEE Int. Conf. Multimedia Computing and Systems*, Hiroshima, Japan, June 1996, pp. 144–150.

[12] M. Wu and W. Shu, "Scheduling for large-scale parallel video servers," in *Proc. Sixth Symp. Frontiers of Massively Parallel Computation*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1996, pp. 126–133.

[13] C. S. Freedman and D. J. DeWitt, "The SPIFFI scalable video-on-demand system," in *Proc. ACM SIGMOD'95*, June 1995, pp. 352–363.

[14] Y. B. Lee and P. C. Wong, "A server array approach for video-on-demand service on local area networks," in *IEEE INFOCOM '96*, San Francisco, CA, Mar. 1996.

[15] S. S. Rao, H. M. Vin, and A. Tarafdar, "Comparative evaluation of server-push and client-pull architectures for multimedia servers," in *Proc. 6th NOSSDAV*, Zushi, Japan, Apr. 1996, pp. 45–48.

[16] E. Biersack, W. Geyer, and C. Bernhardt, "Intra- and inter-stream synchronization for stored multimedia streams," in *Proc. IEEE Int. Conf. Multimedia Computing & Systems*, Hiroshima, Japan, June 17–23, 1996.

[17] C. Bernhardt and E. Biersack, "The server array: A scalable video server architecture," in *High-Speed Networks for Multimedia Applications*. Dordrecht, The Netherlands: Kluwer, 1996.

[18] 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," in *Proc. Sixth Int. Workshop on Network and Operating System Support for Digital Audio and Video*, Zushi, Japan, Apr. 1996.

[19] A. Reddy, "Scheduling and data distribution in a multiprocessor video server," in *Proc. Second IEEE International Conference on Multimedia Computing and Systems*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1995, pp. 256–263.

[20] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Trans. Software Eng.*, vol. 15, pp. 847–853, July 1989.

[21] D. Mills, "Internet time synchronization: The network time protocol," *IEEE Trans. Commun.*, vol. 39, pp. 1482–1493, Oct. 1991.

[22] Z. Yang and T. A. Marsland, Eds., *Global States and Time in Distributed Systems*. Los Alamitos, CA: IEEE Comput. Soc. Press, 1994.

[23] R. Tewari, D. M. Dias, R. Mukherjee, and H. M. Vin, "High availability in clustered multimedia servers," in *Proc. 12th Int. Conf. Data Engineering*, 1996, pp. 645–654.

[24] P. C. Wong and Y. B. Lee, "Redundant array of inexpensive servers (RAIS) for on-demand multimedia services," in *Proc. ICC'97*, Montreal, QC, Canada, June 8–12, 1997.

[25] J. N. Lloyd and K. S. Samuel, *Urn Models and Their Application*. New York: John Wiley, 1997, pp. 125–126.

Jack Y. B. Lee (M'95) is an Associate Professor with the Department of Information Engineering, at the Chinese University of Hong Kong. He directs the Multimedia Communications Laboratory (http://www.mcl.ie.cuhk.edu.hk) and conducts research in distributed multimedia systems, fault-tolerant systems, multicast communications, and Internet computing.