Buffer Management and Dimensioning for a Pull-Based Parallel Video Server

Jack Y. B. Lee

***Abstract*—Recently, there has been a trend toward designing video-on-demand systems using parallel-server architectures. By exploiting server-level parallelism, researchers can break through the performance limit of a single server, while keeping the system cost low by leveraging on commodity hardware platforms. A number of studies have demonstrated the feasibility of building parallel video servers around the client-pull architecture and one can even incorporate data redundancy into the system to sustain server-level failures. However, due to randomness of request arrivals and server processing time, dimensioning the server resource requirement is often difficult. This paper tackles the problem of buffer management and dimensioning for such a pull-based parallel video server. Using a generic buffer-pool model with worst-case analysis, upper bounds on the server buffer requirement are derived for a parallel-server design with multiple disks per server. The obtained bounds are independent of placement policy, video bit-rate, disk-scheduling discipline, and even number of servers in the system, making it applicable to a wide range of server designs. The analytical results also proved that the scalability of this pull-based server design will not be limited by the server buffer requirement.**

***Index Terms*—Buffer management, parallel video server, scalability, server design, system dimensioning, video-on-demand.**

I. Introduction

CONVENTIONAL video-on-demand (VoD) systems are usually designed and implemented around the single-server approach where video titles are stored in a video server for storage, retrieval, and delivery. While the design of such VoD systems are well understood, the capacity constraints (storage and bandwidth) of the server hardware often limit the system to no more than a few hundred users. To go beyond this limit, one either has to use replication [1]–[3] or expensive massively-parallel hardware platforms [4], [5].

Recently, a number of researchers [6]–[19] have begun studies on the design and implementation of scalable video-on-demand (VoD) systems using parallel-server architectures. Interested readers can find a general overview in [20]. The objective in using server-parallelism is to break through the capacity constraint of a single server but without the cost of a full replication across multiple servers. Analogous to disk arrays, most of the studies using parallel-server architectures make use of data striping at the server level to avoid data replication. Specifically, a compressed video title is divided into many stripe units, and these are then distributed to the servers according to a specific placement policy. In this way, all servers will then evenly share the loads from the video clients.

While the idea of parallel-server architecture may appear simple, designing a video server using server-parallelism raises many problems not found in conventional video server designs. In particular, one has to: 1) devise striping and placement algorithms for the storage of video titles among the servers; 2) devise retrieval scheduling algorithms to read video data off the disk array; 3) devise scheduling algorithms to coordinate transmissions from the servers; and 4) devise buffering and presentation algorithms at the client to maintain a smooth video playback, etc. Moreover, these algorithms should be scalable in the sense that the hardware resources required at each server should not increase drastically when scaling up the system (i.e., adding more servers).

While performance studies on push-based parallel video servers have been conducted by a number of researchers [6]–[9], [12], [18], [19], there are very few studies that investigate the performance of pull-based parallel video server designs. In this paper, we tackle the problem of buffer management and dimensioning in a pull-based parallel video server, with particular attention to achieving scalability.

The main contributions of this paper are: 1) we derive upper bounds for the server buffer requirement in a pull-based parallel video server; 2) we show that the bounds are independent of the disk placement policy, the video bit-rate, and the disk-scheduling discipline; 3) we discover that for a multiple-disk server having more than two disks, the buffer requirement will be larger if the server capacity is limited by the network sub-system; and 4) we prove that the per-server buffer requirement is independent of the number of servers in the system and hence will not limit the scalability of the design.

The rest of the paper is organized as follows. Section II gives an overview of the parallel video server architecture; Section III derives upper bounds for single-disk servers. Section IV extends the analysis in Section III to multiple-disk servers. Section V discusses implications of the obtained bounds. Section VI reviews some related works and compares them with our approach. Lastly, Section VII concludes the paper.

II. System Architecture

Fig. 1 depicts the general architecture of the pull-based parallel video server studied in this paper. It is first proposed

Manuscript received October 30, 1998; revised December 22, 2000. This work was supported in part by Grant CUHK6095/99E from the HKSAR Research Grant Council and by the Area-of-Excellence in Information Technology, a research grant from the HKSAR University Grants Council. This paper was recommended by Associate Editor S.-F. Chang.

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

Publisher Item Identifier S 1051-8215(01)03016-6.

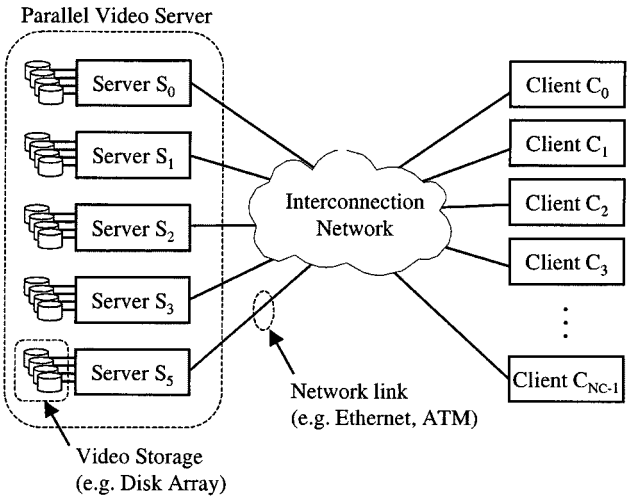

Fig. 1. Architecture of a (five-servers) parallel video server.

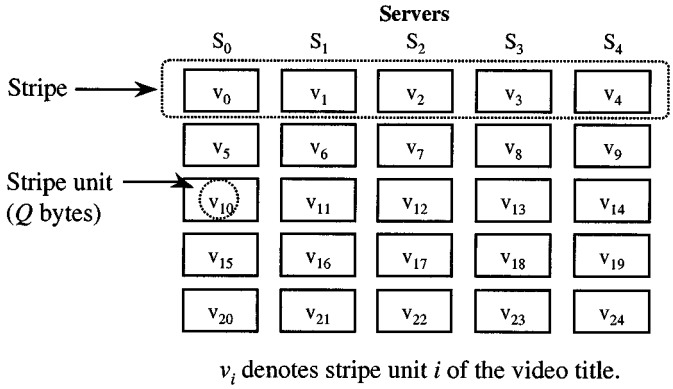

Fig. 2. Data organization in a parallel video server.

in [11] and comprises multiple independent servers connected by an interconnection network. Each video title is divided into fixed-size blocks of Q bytes and distributed to the servers in a round-robin manner as shown in Fig. 2. This fixed-size block striping algorithm is called space striping [20], as opposed to striping in units of video frames, called time striping. Since a stripe unit in space striping is significantly smaller than a video title (kilobytes versus megabytes), this enables fine-grain load sharing (as oppose to course-grain load sharing in data partition) among servers. Moreover, the loads are evenly distributed over all servers regardless of skewness in video retrievals.

A. Service Model

To retrieve video data for playback, a client sends requests to the servers according to the placement policy to retrieve stripe units for playback. Upon receiving a request, a server will read the requested stripe unit off the storage device and transmit it to the client at a controlled data rate. This model of controlling the data flow between the client and the servers is called the client-pull service model [20], as opposed to the server-push service model where the server schedules the retrieval and transmission of video data.

Among the existing studies on parallel video servers, the majority of them employed the server-push service model [4]–[6], [8], [9], [12], [18], [19]. The challenge in push-based parallel video server is inter-server clock-synchronization. As long as the server clocks are synchronized, round-based scheduling algorithms similar to single-server designs can be employed, resulting in relatively simple performance analysis and system dimensioning. By contrast, pull-based parallel video servers do not need clock-synchronization among the servers and hence the servers in the system can be fully autonomous. This eliminates the need for a distributed clock-synchronization protocol, which is nontrivial in its own right.

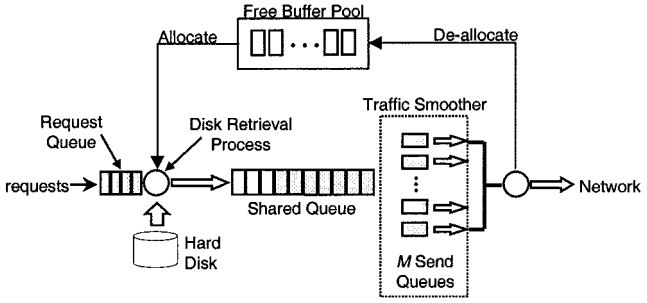

Fig. 3. Design of a server node in the pull-based parallel video server.

B. Server Architecture

Inside each server in the system, there are one request queue, one shared queue, and M send queues, as shown in Fig. 3. All these queues share a pool of buffers allocated by the system at startup. Incoming requests are first stored into the request queue. The disk-retrieval process serves requests in batches by allocating free buffers and reading video blocks off the disk into the buffers. After retrieval, these buffers are then pushed into the shared queue to wait for transmission at the traffic smoother. When a send queue becomes empty, a buffer unit from the shared queue is pushed into the empty send queue for transmission. Each send queue can contain only one buffer unit at a time. After transmission, this buffer unit will be released for reuse. Note that there is no memory copy in passing buffer units among the server queues.

The traffic smoother is introduced to control the bustiness of outgoing traffic at the server to avoid congestion at the network and the client. Specifically, in each round of transmission, the server will transmit one packet from each of the nonempty send queues in a round-robin manner so that video packets destined for different clients are interleaved. For example, with a block size of 64 KB and a packet size of 8 KB (excluding headers), the traffic smoother will completely transmit a video block in eight rounds. To further avoid congestion, we define a minimum round size $M_{\min}$ $(1 \leq M_{\min} \leq M)$, such that the traffic smoother will introduce artificial delay to maintain a minimum round time as if there are $M_{\min}$ nonempty send queues.

It is worth noting that by setting $M = 1$, the traffic smoother reduces to the first-come-first-serve (FCFS) discipline, and by setting M equal to the maximum number of concurrent video streams, the traffic smoother reduces to simple Round Robin. In a parallel-server system, M will usually be set equal to the client-server ratio to maintain a proper transmission rate. Interested readers are referred to [11] for details on this traffic smoother.

C. Buffer-Management Scheme

As fixed-size buffers are used throughout the system, they can be pre-allocated at system initialization to form a buffer pool that will be reused without further invoking memory allocation services from the operating system. This buffer-management scheme has been shown to significantly outperform the general buffer manager provided by the compiler and the operating system [21]. The problem is how many buffers should be allocated so that maximum system performance can be achieved. Clearly, too few buffers will negatively impact system performance due to stalling in either the disk-retrieval process or the traffic smoother. On the other hand, adding buffers beyond a certain limit will not help to further improve system performance.

Note that as a request consumes far less memory than a data block (tens of bytes versus tens of kilobytes), we will ignore the memory needed for storing the incoming requests in the rest of the paper. In the next section, we derive upper bounds for the amount of buffers required to achieve maximum system performance for a single-disk server and extend the analysis in Section IV to obtain similar bounds for multiple-disk server.

Finally, we should emphasize that the obtained bounds only guarantee that the server throughput will not be hampered by insufficient buffers. They do not, for example, guarantee that the load on all servers will be balanced, nor guarantee that video playback at the clients will be continuous. These are separate problems and hence require different solutions (e.g., using admission scheduling to guarantee load balancing [13], and using buffering and prefetching at the client to guarantee playback continuity [11]).

III. Single-Disk Case

Let L_S be the total number of buffers in the buffer pool at the server and Q be the size of a buffer (also the size of a retrieved data block). If L_S is too small, the retrieval process may find that there are insufficient free buffers to start a new service round. In this case, the disk will have to stop serving requests even if there are requests waiting in the request queue, as shown in Fig. 4. On the other hand, Fig. 5 shows that the traffic smoother may need to temporary stop transmission to wait for the disk to retrieve more data blocks. Clearly, the occurrences of these two scenarios depend on the relative speed of the disk-retrieval process, the traffic smoother, and the amount of buffers to pipeline the two processes.

For the disk-retrieval process, most disk-scheduling algorithms (e.g., SCAN and C-SCAN, etc.) serve requests in rounds to reduce disk-seek overhead [22], [23]. Let L_D be the number of requests served in a round, the maximum round time denoted as T_D can then be derived for a particular scheduling scheme. Note that only the worst-case (maximum) round time needs to be considered in system dimensioning because a VoD system must provide performance guarantee.

In general, we cannot assume when a particular request is served in a round. However, we can be sure that all L_D requests will be completely served at the end of a round. The disk throughput is therefore given by

$$S_D = \frac{L_D Q}{T_D}. \tag{1}$$

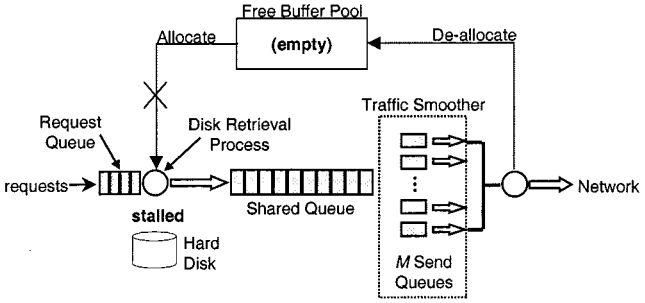

Fig. 4. The disk stalls (i.e., stops serving requests) to wait for the traffic smoother to release buffers.

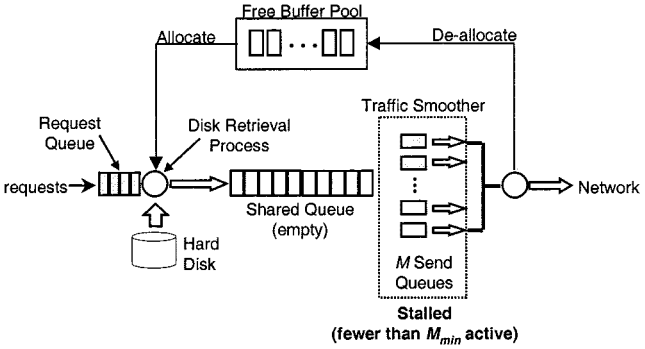

Fig. 5. The traffic smoother stalls (i.e., fewer than $M_{\min}$ send queues active) to wait for the disk to retrieve more data.

Now consider the traffic smoother. We assume that the network interface can send data at a maximum rate of S_N. Then the minimum time T_N required for the network to send $L_D Q$ bytes of data would be given by

$$T_N = \frac{L_D Q}{S_N} \tag{2}$$

or

$$S_N = \frac{L_D Q}{T_N}. \tag{3}$$

Clearly, the server throughput is limited either by the disk (disk bound) or the traffic smoother (network bound), whichever is smaller

$$S_S = \begin{cases} S_N, & \text{if } T_D \le T_N \\ S_D, & \text{otherwise} \end{cases} \tag{4}$$

However, these bounds can only be achieved if there are sufficient buffers to pipeline the disk-retrieval process and the traffic smoother.

Before deriving the number of buffers needed, we first make a few assumptions on the disk-retrieval process. We assume that the disk-retrieval process starts a new round of service only if: 1) there are L_D or more requests waiting for service in the request queue and 2) there are L_D or more free buffers available to store retrieved data. The first assumption models the behavior of round-based disk schedulers (e.g., SCAN, GSS) that batch requests to reduce seek-time overhead. The second assumption is trivial as retrieval cannot proceed without available buffers.

Since we want to ensure that the server can achieve maximum throughput under heavy load, we assume that there will always

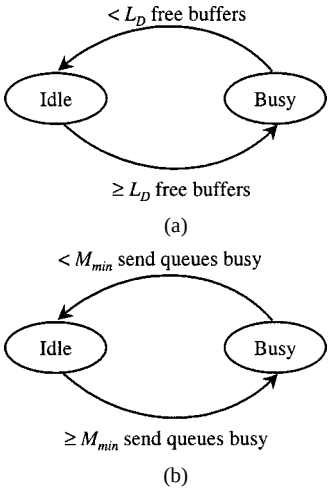

Fig. 6. State transition for the: (a) disk-retrieval process and (b) traffic smoother.

be L_D or more requests waiting at the request queue. That is, the server is always busy with requests pending for service.

Under the previous assumptions, the retrieval process then operates between two states: idle and busy (Fig. 6). The retrieval process enters idle state if there are fewer than L_D free buffers available for the retrieval process to start a new service round. The retrieval process leaves idle state and enters busy state once the two conditions specified in the previous paragraph are satisfied.

A. Server Is Network Bound

We first consider the case where the server is network bound, i.e., $T_D \leq T_N$. In this case, we want to make sure that the traffic smoother (the bottleneck) will never become idle because of insufficient buffers (Fig. 6). We start with the time the retrieval process is in the idle state and determine the number of server buffers needed to ensure that the traffic smoother will always have data for transmission. If the retrieval process is in the idle state, there will be fewer than L_D free buffers in the buffer pool. Let W_S be the total amount of data in all send queues in the traffic smoother, the total amount of data in the system (shared queue + send queues), denoted by W_I, waiting for transmission is given by

$$W_I \geq (L_S - L_D + 1)Q + W_S. \tag{5}$$

From the previous section, we know that the traffic smoother will not send data at full rate if there are fewer than $M_{\min}$ occupied send queues. In the next lemma, we use this observation to define a sufficient condition for the traffic smoother to send data at full rate S_N.

Lemma 1: If there are more than $(M_{\min} - 1)Q$ bytes of data waiting for transmission at the server, then the traffic smoother will be transmitting at full rate S_N.

Proof: Since each buffer can accommodate at most Q bytes of data for transmission, the minimum number of buffers to store $(M_{\min} - 1)Q$ bytes of data is simply $(M_{\min} - 1)$. Hence, to store more than $(M_{\min} - 1)Q$ bytes of data, at least $M_{\min}$ buffers are needed. As all M send queues must be occupied before data can be stored inside buffers in the shared queue, all these $M_{\min}$ buffers must be in the traffic smoother. Therefore the minimum round size (cf. Section II-B) is satisfied and the traffic smoother will be sending data at full rate S_N. ■

Using Lemma 1, we can determine the number of buffers needed to keep the traffic smoother sending at full rate whenever the disk-retrieval process is idle.

Theorem 1: If there are at least $(L_D + M_{\min} - 2)$ buffers, then the traffic smoother will always be transmitting at full rate whenever the disk-retrieval process is in the idle state.

Proof: From Lemma 1, to ensure that the traffic smoother will be sending data at full rate whenever the disk-retrieval process is idle, we need

$$W_I > (M_{\min} - 1)Q. \tag{6}$$

Substituting (5) into W_I, we then have

$$(L_S - L_D + 1)Q + W_S > (M_{\min} - 1)Q. \tag{7}$$

Since this must be true for any valid value of W_S and $0 < W_S \leq MQ$, we have

$$L_S \geq L_D + M_{\min} - 2 \tag{8}$$

and the result follows. ■

Equation (8) gives the number of buffers needed at the server to ensure that the traffic smoother can transmit at full rate when the retrieval process is in the idle state. Next, we consider the case when the disk-retrieval process leaves the idle state and enters the busy state. We derive in the next lemma the minimum amount of data in the system at this instant.

Lemma 2: At the time instant the disk-retrieval process leaves the idle state and enters the busy state, the minimum amount of data waiting for transmission in the system is at least

$$W_{IB} \geq (L_S - L_D - M + 1)Q. \tag{9}$$

Proof: Just before the disk-retrieval process leaves the idle state and enters the busy state, there are at most $(L_D - 1)$ free buffers. Since the traffic smoother can free at most M free buffers at a time instant, the number of free buffers is at most $(L_D - 1 + M)$ when the disk leaves the idle state and enters the busy state. As the retrieval process uses L_D of these free buffers for the new service round, this leaves at most $(M - 1)$ free buffers in the system. Note that free buffers and buffers within the retrieval process do not contain data ready for transmission, hence the amount of data in the system waiting for transmission must be at least $L_S Q - (M - 1)Q - L_D Q$ and the result follows. ■

Next, we derive the condition to keep the traffic smoother sending at full rate at any time t after the disk-retrieval process has left the idle state and entered the busy state. We first prove a lemma bounding the maximum difference between the amount of data retrieved and the amount of data transmitted in the time interval t.

Lemma 3: Let $a(t)$ be the amount of data transmitted, and $b(t)$ be the amount of data retrieved in a time interval t after the disk-retrieval process entered the busy state. If the disk-retrieval

process does not become idle again during the time interval t, then the difference between $a(t)$ and $b(t)$ is at most

$$a(t) - b(t) \leq L_D Q \quad \forall t \geq 0. \tag{10}$$

Proof: Since the maximum transmission rate of the traffic smoother is S_N, the maximum amount of data that can be transmitted in a time interval t is

$$a(t) \leq tS_N. \tag{11}$$

Substituting S_N with (3) gives

$$a(t) \leq t\frac{L_D Q}{T_N}. \tag{12}$$

For the disk-retrieval process, the minimum number of service rounds that can be completed in a time interval t is $\lfloor t/T_D \rfloor$ since the disk-retrieval process starts a new round when it leaves the idle state and enters the busy state. Therefore, the minimum amount of data that can be retrieved is at least

$$b(t) \geq \left\lfloor \frac{t}{T_D} \right\rfloor L_D Q. \tag{13}$$

Therefore, the difference between $a(t)$ and $b(t)$ is

$$a(t) - b(t) \leq \left(\frac{t}{T_N} - \left\lfloor \frac{t}{T_D} \right\rfloor \right) L_D Q. \tag{14}$$

Now, as $T_N \geq T_D$, we can replace T_N by T_D to obtain

$$a(t) - b(t) \leq \left(\frac{t}{T_D} - \left\lfloor \frac{t}{T_D} \right\rfloor \right) L_D Q. \tag{15}$$

Since $(x - \lfloor x \rfloor) < 1$ for all positive real numbers x, (15) is bounded by

$$a(t) - b(t) < L_D Q \tag{16}$$

and the result follows. ∎

Using Lemmas 2 and 3, we can determine the maximum number of buffers required to keep the traffic smoother transmitting data at full rate when the disk is in the busy state.

Theorem 2: If the total number of buffers L_S is at least $(2L_D + M + M_{\min} - 2)$, then the traffic smoother will be transmitting at full rate whenever the disk-retrieval process is in the busy state.

Proof: To determine the condition for keeping the traffic smoother transmitting at full rate at any time when the disk-retrieval process is not idle, we consider a time t after the disk-retrieval process left the idle state and entered the busy state. We use $W_B(t)$ to denote the amount of data in the system at a time t after the disk-retrieval process left the idle state and entered the busy state. We can express $W_B(t)$ as

$$W_B(t) \geq W_{IB} - a(t) + b(t). \tag{17}$$

Substituting (9) from Lemma 2 and (16) from Lemma 3 into (17), we have

$$W_B(t) > (L_S - L_D - M + 1)Q - L_D Q. \tag{18}$$

To ensure that the traffic smoother transmits at full rate, we need to ensure that (cf. Lemma 1)

$$W_B(t) > (M_{\min} - 1)Q \quad \forall t \geq 0. \tag{19}$$

Hence, we have

$$(L_S - L_D - M + 1)Q - L_D Q \geq (M_{\min} - 1)Q. \tag{20}$$

Rearranging, we can solve for L_S as

$$L_S \geq 2L_D + M + M_{\min} - 2 \tag{21}$$

and the result follows. ∎

Theorem 2 states that as long as the retrieval process is in the busy state, the traffic smoother will always have data to transmit if $L_S \geq (2L_D + M + M_{\min} - 2)$. Note that this buffer requirement also satisfies Theorem 1 for the idle state. Therefore, Theorem 2 bounds the number of buffers required to avoid traffic smoother stalling.

B. Server Is Disk Bound

If $T_D > T_N$, then the server is disk bound. Since the disk becomes the bottleneck, we want to ensure that the disk-retrieval process will never be blocked from serving waiting requests due to insufficient buffers. As the disk-retrieval process will not be able to start a new service round (i.e., stalled) if there are fewer than L_D free buffers in the system, our objective is to determine the number of buffers needed to guarantee that there are always L_D or more free buffers in the system.

We define that the traffic smoother is in the idle state if less than $M_{\min}$ send queues are active. In this case, artificial delay will be added (Section II-B) and the aggregate transmission rate will be less than the network throughput S_N. Let U_S be the total amount of data in all send queues in the traffic smoother, and U_Q be the total amount of data in the shared queue. Clearly, U_Q will be zero and $U_S \leq (M_{\min} - 1)Q$ (cf. Lemma 1) if the traffic smoother is in the idle state. To guarantee that the disk-retrieval process does not stall, we must ensure that at least L_D free buffers are available, i.e.,

$$L_S - L_D - (M_{\min} - 1) \geq L_D. \tag{22}$$

Rearranging gives the buffer requirement for the idle state

$$L_S \geq 2L_D + (M_{\min} - 1) \tag{23}$$

Now consider the time instant when the traffic smoother transits from the idle state to the busy state, where $M_{\min}$ or more send queues are active. Since there is no data at the shared queue during the idle state, the state transition must be due to completion of a service round at the disk-retrieval process. Theorem 3 below sets the condition for guaranteeing continuous disk retrieval at this time instant.

Theorem 3: If the total number of buffers L_S is at least $(2L_D + (M_{\min} - 1))$, then the disk-retrieval process will not be stalled due to lack of free buffers at the time instant the traffic smoother transits from the idle state to the busy state.

Proof: Just before the traffic smoother leaves the idle state and enters the busy state, there are at most $(M_{\min} - 1)$ occupied buffers (cf. Lemma 1). Since the disk-retrieval process can

retrieve at most L_D buffers of video data at a time, the number of occupied buffers when the traffic smoother leaves the idle state and enters the busy state is at most $(L_D + M_{\min} - 1)$. To avoid stalling the disk-retrieval process, we need to ensure that the system still has at least L_D available free buffers. Hence the buffer requirement is given by $(L_D + M_{\min} - 1) + L_D$ and the result follows. ■

To further derive the number of buffers needed to keep the disk-retrieval process from stalling after the traffic smoother entered the busy state, we first need to derive the maximum amount of data in the system in the next lemma.

Lemma 4: At the instant the traffic smoother leaves the idle state and enters the busy state, the amount of data waiting for transmission in the system is at most

$$W_{IB} \leq (L_D + M_{\min} - 1)Q. \tag{24}$$

Proof: Similar to the proof of Theorem 3. ■

Next we derive a bound for the difference between the amount of data retrieved and the amount of data transmitted in a time t after the traffic smoother entered the busy state.

Lemma 5: Let $a(t)$ be the amount of data transmitted, and $b(t)$ be the amount of data retrieved in a time interval t after the traffic smoother entered the busy state. If the traffic smoother does not become idle again during the time interval t, then the difference between $b(t)$ and $a(t)$ is at most

$$b(t) - a(t) < L_D Q \quad \forall t \geq 0. \tag{25}$$

Proof: As the traffic smoother is in the busy state (i.e., $M_{\min}$ or more active send queues), it transmits video data at the full effective network throughput S_N. Hence, the amount of data that can be transmitted in a time interval t is just given by

$$a(t) = tS_N. \tag{26}$$

Substituting S_N with (3) gives

$$a(t) = t\frac{L_D Q}{T_N}. \tag{27}$$

For the disk-retrieval process, the maximum number of service round completions that can occur in a time interval t is $(\lfloor t/T_D \rfloor + 1)$. Since each service round completion retrieves $L_D Q$ bytes of data from the disk, the amount of data that can be retrieved is at most

$$b(t) \leq \left(\left\lfloor \frac{t}{T_D} \right\rfloor + 1 \right) L_D Q. \tag{28}$$

Therefore, the difference between $b(t)$ and $a(t)$ is bounded by

$$b(t) - a(t) \leq \left(\left\lfloor \frac{t}{T_D} \right\rfloor + 1 - \frac{t}{T_N} \right) L_D Q. \tag{29}$$

Now, as $T_N < T_D$, we can replace T_N by T_D in (29) to obtain

$$b(t) - a(t) < \left(\left\lfloor \frac{t}{T_D} \right\rfloor - \frac{t}{T_D} + 1 \right) L_D Q. \tag{30}$$

Since $(\lfloor x \rfloor - x) \leq 0$ for all positive real numbers x, (30) must be bounded by

$$b(t) - a(t) < L_D Q \tag{31}$$

and the result follows. ■

Using Lemmas 4 and 5, we can determine the number of buffers required to guarantee that the disk-retrieval process will not be stalled due to lack of free buffers when the traffic smoother is in the busy state.

Theorem 4: If the total amount of buffer L_S is at least $(3L_D + M + M_{\min} - 2)$, then the disk-retrieval process will never be stalled due to lack of free buffers when the traffic smoother is in the busy state.

Proof: We use $W_B(t)$ to denote the amount of data in the system at a time t after the traffic smoother left the idle state and entered the busy state. We can express $W_B(t)$ as

$$W_B(t) = W_{IB} - a(t) + b(t). \tag{32}$$

Substituting (24) from Lemma 4 and (25) from Lemma 5 into (32), we have

$$W_B(t) < (L_D + M_{\min} - 1)Q + L_D Q. \tag{33}$$

To store this amount of data, we need at least $(2L_D + M_{\min} - 1)$ buffers but no more than $(2L_D + M_{\min} + M - 2)$ because only the M send queues in the traffic smoother can be partially occupied. Hence, to ensure that the disk-retrieval process will not stall due to lack of free buffers, we just need to ensure that

$$L_S - (2L_D + M_{\min} + M - 2) \geq L_D. \tag{34}$$

Rearranging, we can solve for L_S as

$$L_S \geq 3L_D + M + M_{\min} - 2 \tag{35}$$

and the result follows. ■

So far, we have assumed that the disk-retrieval process is working under worst-case scenario, i.e., the disk service round is at the maximum of T_D seconds. Given the number of buffers as derived in Theorem 4, the disk-retrieval process will never be stalled due to lack of free buffer, and hence guarantees a throughput of S_D bytes/s, which is also the maximum achievable server throughput. However, if the retrieval pattern is not worst case (e.g., clustered around a small disk region), then the disk-retrieval process could still be stalled momentarily due to reduction in seek time. The next theorem shows that the disk throughput in this case is still at least S_D.

Theorem 5: The disk-retrieval process can achieve a throughput of at least S_D, even if it stalls under nonworst-case scenarios.

Proof: Assume that the retrieval pattern in a service round, say round k, is nonworst-case in the sense that the length of the service round, say τ, is less than T_D. Under this scenario, it is possible that the traffic smoother cannot free enough buffers for the disk-retrieval process to start service round $k+1$, i.e., stalled. Let v_k be the number of free buffers at the start of service round k. As the disk-retrieval process stalls in round $k+1$, clearly $0 \leq v_k < L_D$. To enable the disk-retrieval process to start round $k+1$, the traffic smoother needs to clear at most L_D free buffers.

Now to free L_D buffers, the traffic smoother needs to transmit at most $L_D Q$ bytes of data. Since the traffic smoother is in busy state when the disk-retrieval process stalls, the time it takes to transmit $L_D Q$ bytes of data is simply T_N seconds. But $T_D >$

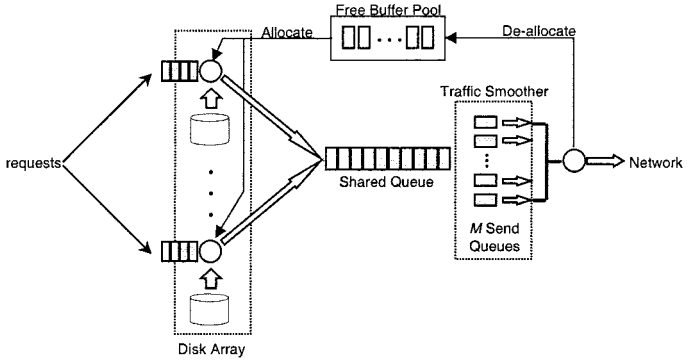

Fig. 7. Model for a multiple-disk server node in the parallel video server.

T_N. Hence, the time interval between service round k and $k+1$ will be smaller than T_D. Therefore, the disk throughput for round k becomes $L_DQ/T_N > L_DQ/T_D = S_D$ and the result follows. ■

IV. Multiple-Disk Case

Section III derives the server buffer requirement for a single-disk server. In practice, it is likely that multiple disks will be used to fully utilize the I/O channels, CPU capacity, as well as the network interface throughput. In this section, we extend the derivations in Section III to derive the server buffer requirement for multi-disk configurations.

Fig. 7 depicts the model for a multiple-disk server node. Let N_D be the number of disks in the server node, each having a separate request queue for queueing incoming requests. A video stream is divided into blocks of Q bytes and distributed to the disks according to a striping algorithm. For generality, we do not assume the use of any specific striping algorithm. The only assumption is that the transmission scheduling and striping algorithm is designed in such a way that a request retrieves one video block of Q bytes from one of the N_D disks.

When a request arrives at the server node, it is immediately dispatched to one of the request queues. On the other hand, retrieved data blocks from the disks will all be pushed into the single shared queue to wait for transmission. We define that the disk-retrieval process is in the idle state if any one of the disks is in the idle state. Otherwise, the disk-retrieval process is in the busy state. Under this new model, we need to modify the definitions for disk- and network-bound conditions. Specifically, we define the system to be network bound if $N_DT_N \geq T_D$, and disk bound otherwise. In the following, we consider each case in turn.

A. Server Is Network Bound

The server is network bound implies that $N_DT_N \geq T_D$. If the retrieval process is in the idle state, then there will be at most $L_D - 1$ empty buffer inside the shared queue. Let n $(1 \leq n \leq N_D)$ out of the N_D disks are idle and let W_S be the total amount of data in all send queues in the traffic smoother. Then the total amount of data W_I in the system waiting for transmission is given by

$$W_I \geq (L_S - L_D + 1 - (N_D - n)L_D)Q + W_S. \quad (36)$$

This is because $(N_D - n)$ disks will be in the busy state, each occupying L_D buffers for the ongoing service round and these buffers are not yet ready for transmission.

Using Lemma 1 and (36), we can derive the buffer requirement needed to keep the traffic smoother sending at full rate whenever the disk-retrieval process is idle.

Theorem 6: If there are at least $(N_DL_D + M_{\min} - 2)$ buffers, then the traffic smoother will always be transmitting at full rate whenever the disk-retrieval process is in the idle state.

Proof: From Lemma 1, to ensure that the traffic smoother will be sending data at full rate, we need

$$W_I > (M_{\min} - 1)Q. \quad (37)$$

Substituting (36) for W_I, we then have

$$(L_S - L_D + 1 - (N_D - n)L_D)Q + W_S > (M_{\min} - 1)Q. \quad (38)$$

Since this must be true for any valid value of W_S where $0 < W_S \leq MQ$, and any valid value of n where $1 \leq n \leq N_D$, we can obtain a bound for L_S from

$$L_S \geq N_DL_D + M_{\min} - 2 \quad (39)$$

and the result follows. ■

Theorem 6 bounds the number of buffers needed to ensure that the traffic smoother will transmit at full rate when the retrieval process is in the idle state. Next, we consider the instant when the disk-retrieval process leaves the idle state and enters the busy state. In the next lemma, we first derive the minimum amount of data in the system at the state-transition instant.

Lemma 6: When the disk-retrieval process leaves the idle state and enters the busy state, the amount of data waiting for transmission in the system is at least

$$W_{IB} \geq (L_S - N_DL_D - M + 1)Q. \quad (40)$$

Proof: Just before the disk-retrieval process leaves the idle state and enters the busy state, there are at most $(L_D - 1)$ free buffers. Since the traffic smoother can free at most M free buffers at one time, the number of free buffers when the disk leaves the idle state and enters the busy state is at most $(L_D - 1 + M)$. Now the retrieval process uses at least L_D of these free buffers for the new service round, and hence leaves at most $(M - 1)$ free buffers in the system. Since all disks are busy serving requests in the busy state, a total of N_DL_D buffers are used by the retrieval process. Noting that free buffers and buffers within the retrieval process do not contain data ready for transmission, the amount of data in the system waiting for transmission must be at least $L_SQ - (M - 1)Q - N_DL_DQ$ and the result follows. ■

Next we derive the condition to keep the traffic smoother sending at full rate at any time t after the disk-retrieval process has left the idle state and entered the busy state. We first prove a lemma bounding the maximum difference between the amount of data retrieved and the amount of data transmitted in the time interval t.

Lemma 7: Let $a(t)$ be the amount of data transmitted, and $b(t)$ be the amount of data retrieved in a time interval t after the disk-retrieval process entered the busy state. If the disk-retrieval process does not become idle again during the time interval t, then the difference between $a(t)$ and $b(t)$ is bounded by

$$a(t) - b(t) < N_D L_D Q \quad \forall t \geq 0. \tag{41}$$

Proof: Since the maximum transmission rate of the traffic smoother is S_N, the maximum amount of data that can be transmitted in a time interval t is

$$a(t) \leq tS_N. \tag{42}$$

Replacing S_N using (3) gives

$$a(t) \leq \frac{tL_D Q}{T_N}. \tag{43}$$

For the disk-retrieval process, the minimum number of service rounds that can be completed in a time interval t is $N_D \lfloor t/T_D \rfloor$. Therefore, the amount of data that can be retrieved is at least

$$b(t) \geq \left\lfloor \frac{t}{T_D} \right\rfloor N_D L_D Q. \tag{44}$$

The difference between $a(t)$ and $b(t)$ is given by

$$a(t) - b(t) \leq \left(\frac{t}{T_N} - N_D \left\lfloor \frac{t}{T_D} \right\rfloor \right) L_D Q. \tag{45}$$

Now, as $N_D T_N \geq T_D$, we can replace T_N by T_D to obtain

$$a(t) - b(t) \leq \left(N_D \frac{t}{T_D} - N_D \left\lfloor \frac{t}{T_D} \right\rfloor \right) L_D Q. \tag{46}$$

Note that since $(x - \lfloor x \rfloor) < 1$ for all positive real number x, (46) must be bounded by

$$a(t) - b(t) < N_D L_D Q \tag{47}$$

and the result follows. ■

Using Lemmas 6 and 7, we can then determine the number of buffers required to keep the traffic smoother transmitting data at full rate at any time after the disk-retrieval process starts its busy state.

Theorem 7: If there are at least $(2N_D L_D + M + M_{\min} - 2)$ buffers, then the traffic smoother will be transmitting at full rate whenever the disk-retrieval process is in the busy state.

Proof: To determine the condition to keep the traffic smoother at full rate at any time when the disk-retrieval process is not idle, we consider a time t after the disk-retrieval process has left the idle state and entered the busy state. We use $W_B(t)$ to denote the amount of data in the system at a time t after the disk-retrieval process has left the idle state and entered the busy state. We can express $W_B(t)$ as

$$W_B(t) \geq W_{IB} - a(t) + b(t). \tag{48}$$

Substituting (40) from Lemma 6 and (47) from Lemma 7 into (48), we have

$$W_B(t) \geq (L_S - N_D L_D - M + 1)Q - N_D L_D Q. \tag{49}$$

To ensure that the traffic smoother transmits at full rate, we need to ensure that

$$W_B(t) > (M_{\min} - 1)Q \quad \forall t \geq 0 \tag{50}$$

according to Lemma 1. Substituting (49) into $W_B(t)$, we then obtain

$$(L_S - N_D L_D - M + 1)Q - N_D L_D Q \geq (M_{\min} - 1)Q. \tag{51}$$

Rearranging, we can finally solve for L_S as

$$L_S \geq 2N_D L_D + M + M_{\min} - 2 \tag{52}$$

and the result follows. ■

Theorem 7 states that as long as the retrieval process is in the busy state, the traffic smoother will then always have data to transmit if $L_S \geq (2N_D L_D + M + M_{\min} - 2)$. As the buffer requirement for the busy state also exceeds the requirement for the idle state, the traffic smoother will never be stalled by lack of free buffers and hence the system can achieve the maximum throughput of S_N.

B. Server Is Disk Bound

If $T_D > N_D T_N$, then the server is disk bound. Since the disk is now the bottleneck, we want to ensure that the disk-retrieval process will never be blocked from serving waiting requests due to lack of free buffers. As the disk-retrieval process cannot start a new service round unless there are L_D or more free buffers in the system, our objective is to determine the total number of buffers needed to guarantee that there are always L_D or more free buffers.

Again, let U_S be the total amount of data in all send queues in the traffic smoother, and U_Q be the total amount of data in the shared queue. If the traffic smoother is in the idle state, then U_Q will be zero and $U_S \leq (M_{\min} - 1)Q$ (cf. Lemma 1). To guarantee that the disk-retrieval process does not stall in this state, we must ensure that

$$L_S - N_D L_D - (M_{\min} - 1) \geq L_D \tag{53}$$

because the disk-retrieval process consumes at most $N_D L_D$ buffers and the traffic smoother consumes at most $(M_{\min} - 1)$ buffers. Rearranging gives the buffer requirement for the idle state

$$L_S \geq (N_D + 1)L_D + (M_{\min} - 1). \tag{54}$$

Now consider the time instant when the traffic smoother transits from the idle state to the busy state, where $M_{\min}$ or more send queues become active. Since there is no data at the shared queue in the idle state, the state transition must be triggered by completion of a service round at the disk-retrieval process. Note that while there are multiple disks in the system, we assume that service round completions in different disks only occur sequentially. To guarantee continuous disk retrieval at this time instant, we need the following.

Theorem 8: If the total number of buffers L_S is at least $((N_D + 1)L_D + (M_{\min} - 1))$, then the disk-retrieval process will not be stalled due to lack of free buffers at the time instant

the traffic smoother changes from the idle state to the busy state.

Proof: Similar to Theorem 3. ■

To determine the amount of buffer needed to sustain continuous disk retrieval once the traffic smoother entered the busy state, we first consider the difference between the amount of data retrieved and the amount of data transmitted.

Lemma 8: Let $a(t)$ be the amount of data transmitted, and $b(t)$ be the amount of data retrieved in a time interval t after the traffic smoother entered the busy state. If the traffic smoother does not become idle again during the time interval t, then the difference between $b(t)$ and $a(t)$ is at most

$$b(t) - a(t) < N_D L_D Q \quad \forall t \geq 0. \tag{56}$$

Proof: As the traffic smoother is in the busy state (i.e., $M_{\min}$ or more active send queues), it transmits video data at the full effective network throughput S_N. Hence, the amount of data that can be transmitted in a time interval t is just given by

$$a(t) = tS_N. \tag{57}$$

Substituting S_N using (3) gives

$$a(t) = t\frac{L_D Q}{T_N}. \tag{58}$$

For the disk-retrieval process, the maximum number of service round completions that can occur in a time interval t is $N_D(\lfloor t/T_D \rfloor + 1)$ because there are now N_D disks. Since each service round completion retrieves $L_D Q$ bytes of data from the disk, the amount of data that can be retrieved is at most

$$b(t) \leq \left(\left\lfloor \frac{t}{T_D} \right\rfloor + 1 \right) N_D L_D Q. \tag{59}$$

Therefore, the difference between $b(t)$ and $a(t)$ is

$$b(t) - a(t) \leq \left(\left\lfloor \frac{t}{T_D} \right\rfloor + 1 - \frac{t}{N_D T_N} \right) N_D L_D Q. \tag{60}$$

Now, since $N_D T_N < T_D$, we can replace T_N by T_D to obtain

$$b(t) - a(t) < \left(\left\lfloor \frac{t}{T_D} \right\rfloor - \frac{t}{T_D} + 1 \right) N_D L_D Q. \tag{61}$$

Note that $(\lfloor x \rfloor - x) \leq 0$ for all positive real number x, (61) must be bounded by

$$b(t) - a(t) < N_D L_D Q \tag{62}$$

and the result follows. ■

Using Lemmas 4 and 8, we can determine the number of buffers required to guarantee that the disk-retrieval process will not be stalled due to lack of free buffers when the traffic smoother is in the busy state.

Theorem 9: If the total amount of buffer L_S is at least $((N_D + 2)L_D + M + M_{\min} - 2)$, then the disk-retrieval process will never be stalled due to lack of free buffers when the traffic smoother is in the busy state.

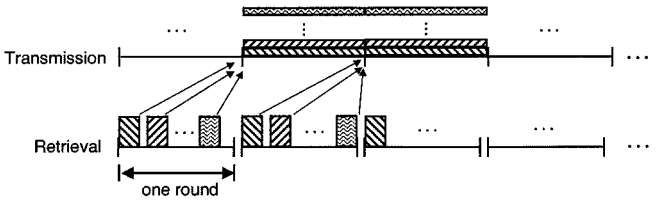

Fig. 8. Double buffering scheme for push-based single-disk video server.

Proof: We use $W_B(t)$ to denote the amount of data in the system at a time t after the traffic smoother left the idle state and entered the busy state. We can express $W_B(t)$ as

$$W_B(t) = W_{IB} - a(t) + b(t). \tag{63}$$

Substituting (24) from Lemma 4 and (62) from Lemma 8 into (63), we have

$$W_B(t) < (L_D + M_{\min} - 1)Q + N_D L_D Q. \tag{64}$$

To store this amount of data, we need at least $((N_D + 1)L_D + M_{\min} - 1)$ buffers but no more than $((N_D+1)L_D + M_{\min} + M - 2)$ because only the M send queues in the traffic smoother can be partially occupied. Hence, to ensure that the disk-retrieval process will not stall due to lack of free buffers, we need to ensure that

$$L_S - ((N_D + 1)L_D + M_{\min} + M - 2) \geq L_D. \tag{65}$$

Rearranging, we can solve for L_S as

$$L_S \geq (N_D + 2)L_D + M + M_{\min} - 2 \tag{66}$$

and the result follows. ■

Similar to the single-disk case, the next theorem shows that the disk throughput is still at least S_D under nonworst-case scenarios.

Theorem 10: The disk-retrieval process can achieve a throughput of at least S_D even if it stalls under nonworst-case scenarios.

Proof: Assume that the retrieval pattern in a service round, say round k in disk j, is less than the worst case and consequently the length of the service round, say τ, becomes less than T_D. Under this scenario, it is possible that the traffic smoother cannot free enough buffers for the particular disk to start service round $k+1$, i.e., stalled. More disks can also be stalled if they also complete their service round earlier. Now as the traffic smoother transmits at the full rate of S_N under this scenario, it can free up at least L_D buffers in a time interval of T_N. Note that the time between disk j starts round k to the expected time for it to start round $k+1$ under worst-case scenario is just T_D. As $T_D > N_D T_N$, the traffic smoother will have freed at least $N_D L_D$ buffers. In other words, we can guarantee that none of the N_D disks will be stalled by the expected time disk j starts round $k+1$ under the worst-case scenario. Since this true for all j and k, the disk-retrieval process will never be stalled and the result follows. ■

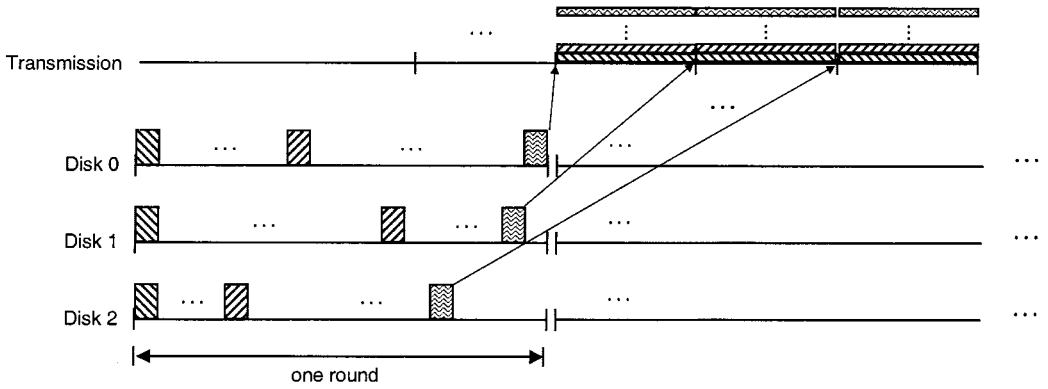

Fig. 9. Extension of double-buffering scheme for push-based, multiple-disk video server.

V. Discussions

As we have not assumed any specific disk-scheduling discipline in the derivations in Sections III and IV, the obtained bounds are valid for any disk-scheduling discipline as long as it serves requests in batches of L_D requests. For example, one can employ any round-based scheduling algorithms such as SCAN, C-SCAN, etc. as the disk scheduler. The corresponding worst-case round length T_D can then be derived using worst-case analysis. It is worth noting that T_D is needed only for determining whether the server is disk bound or network bound. Hence, by taking the maximum of the two, one can always obtain a valid upper bound regardless of T_D.

Another observation is that for servers with multiple disks, the buffer requirement is likely to be larger if the server is network bound. Specifically, since the buffer requirement is equal to $(2N_DL_D+M+M_{\min}-2)$ for the network-bound case and equal to $((N_D+2)L_D+M+M_{\min}-2)$ for the disk bound case, we have

$$\begin{aligned}&(2N_DL_D+M+M_{\min}-2)\\&\quad-((N_D+2)L_D+M+M_{\min}-2)\\&\quad=(N_D-2)L_D>0,\quad \text{for } N_D>2.\end{aligned}\tag{67}$$

Therefore, for more than two disks per server, the buffer requirement will be larger if the server is network bound. Interestingly, if one artificially reduce the disk throughput (e.g., by adding artificial delay to make the round length T_D slightly larger than T_N) to turn the system from network bound to disk bound, the buffer requirement can be reduced. This could be significant for high-capacity servers with a large number of disks (i.e., N_D large).

On the other hand, the derived bounds are also independent of the number of servers in the system. In other words, server buffer requirement in this pull-based parallel video server will not become a limiting factor to the system's scalability. This is in sharp contrast to many buffer-management algorithms employed in push-based parallel video servers (see Section VI-B), in which the per-server buffer requirement increases as the system is scaled up.

VI. Related Works

There are a large body of works on VoD system designs and implementations. In this section, we compare the existing results on buffer management and dimensioning with the approach studied in this paper.

A. Single-Server Architectures

For single-disk, single-node, push-based video servers, the most commonly used buffer-management scheme is double buffering, i.e., two buffers are used to pipeline the retrieval process and the transmission process as shown in Fig. 8. Hence, if a disk round retrieves up to L_D video blocks for L_D concurrent video streams, the buffer requirement will be $2L_D$.

For multi-disks video servers, extending this double buffering scheme could cause scalability problem [23]. Specifically, with N_D disks in the server where each disk service round retrieves one video block for each video stream (see Fig. 9), the buffer requirement per disk would become $2N_DL_D$. Under this scheme, the per-disk buffer requirement increases with the scale of the system. To solve this problem, one can stagger the disk schedules, as shown in Fig. 10 to avoid unnecessary buffer-holding time. This reduces the buffer requirement to $(N_D+1)N_DL_D$ buffers per disk [23]. To further reduce the buffer requirement, the video streams can be divided into multiple groups, with the groups served sequentially in different disk cycles (Fig. 11). This split-schedule scheme can reduce the per-disk buffer requirement to only $2L_D$ buffers, i.e., independent of the number of disks in the system [23].

For the pull-based design studied in this paper, the buffer requirement for the network-bound case is $(2N_DL_D+M+M_{\min}-2)$. Compared with the split-schedule scheme

$$\begin{aligned}&(2N_DL_D+M+M_{\min}-2)-2N_DL_D\\&\quad=M+M_{\min}-2\geq 0.\end{aligned}\tag{68}$$

Since $M\geq M_{\min}\geq 1$, the pull-based design requires at least as many buffers as the split-schedule scheme. However, in practice, M and $M_{\min}$ are usually proportional to the number of video streams to support, hence the difference will increase for servers with more disks.

On the other hand, if the server is disk bound, the buffer requirement will become $((N_D+2)L_D+M+M_{\min}-2)$. Again, compared with the split-schedule scheme

$$\begin{aligned}&((N_D+2)L_D+M+M_{\min}-2)-2N_DL_D\\&\quad=(2-N_D)L_D+M+M_{\min}-2.\end{aligned}\tag{69}$$

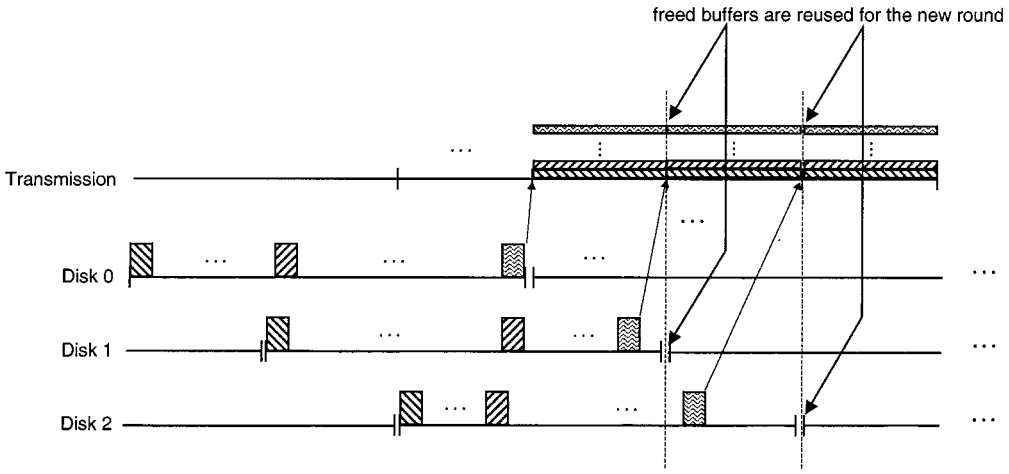

Fig. 10. Offset-schedule scheme for push-based, multiple-disk video server.

In this case, the pull-based design could require fewer buffers than split-schedule for large N_D and small M and $M_{\min}$. However, if we set $M = N_D L_D$ (which is the case in [11]), then the pull-based design will require more buffers.

The previous results show that the pull-based design generally requires more buffers than good push-based designs (e.g., split schedule). This is expected as the server-push service model is centralized, and the disk-retrieval process is periodic. For example, in the system implementation in [11], the authors used $M = 10$, $M_{\min} = 2$, $L_D = 10$, $Q = 64$ KB, and hence the server needs 1.875 MB if it is network bound, and 2.5 MB if it is disk bound. The same hardware would require 1.25 MB for the split-schedule server-push design.

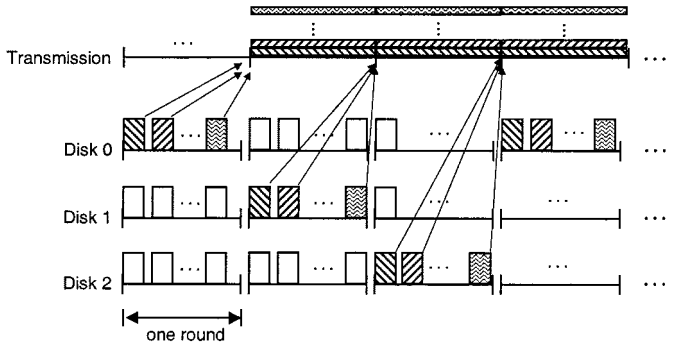

Fig. 11. Split-schedule scheme for push-based, multiple-disk video server.

B. Parallel-Server Architectures

While server buffer management and dimensioning has been studied extensively for single-server, push-based video servers, only a few recent studies [9], [10], [12], [16] have investigated the corresponding issue in parallel push-based video servers. Other studies such as [8], [11], [13]–[15], [17]–[19] focused on other system issues and did not consider buffer management and dimensioning in detail.

The studies by Tewari *et al.* [16] investigated a two-tiered architecture where video data blocks are stored in multiple storage nodes connected to multiple delivery nodes by a high-speed interconnect. A client connects to one of the delivery nodes, which in turn prefetches video data from the storage nodes and then transmits to the client at a controlled data rate. Their study considered the buffer requirement at the delivery nodes and showed that the buffer requirement is linearly proportional to the number of video clients served by the delivery node. Their study did not consider buffer requirement at the storage nodes. Another study by Buddhikot *et al.* [9] employed a custom-designed high-speed ATM interconnect to wire up and synchronize all servers. Their design is push-based and requires one buffer per stream per storage node. In the study by Lee [12] using another push-based architecture, the server buffer requirement is also linearly proportional to the scale of the system.

The main problem with the previous designs is that the per-server buffer requirement increases when scaling up the system by adding more servers. Furthermore, existing server nodes may have to be upgraded with more memory when adding more servers to the system. Finally, the maximum server memory size will limit the ultimate scale of the system. Unlike these push-based architectures, the buffer requirement in the pull-based parallel video server studied in this paper is constant regardless of the number of servers (i.e., scale) in the system. Therefore the server memory constraint will not limit the scalability of the system.

Another study by Freedman *et al.* in [10] investigated a parallel-server architecture with semi-client-pull service model with predictive prefetching at the server nodes. They used simulation to evaluate various algorithms in managing the buffer pool at the server as well as the effect of other system parameters (e.g., stripe size, disk-scheduling alg orithm, etc.). Their simulation results show that a four-node configuration with four disks per node, 512-KB stripe size, and elevator seeking serving 220 video streams requires 128-MB buffer memory per server. For comparison, to support similar number of video streams in a four-node, four-disk parallel video server using the buffer-management scheme studied in this paper, we could set $L_D = 14$, and $M = M_{\min} = 14$. With 512-KB stripe size, the derived buffer requirement is only 69 MB for the network-bound case, and only 55 MB for the disk-bound case.

VII. Conclusion

In this paper, we have tackled the buffer-management and dimensioning problem for a pull-based parallel video server and established upper bounds for the server buffer requirement under network-bound and disk-bound scenarios. The obtained bounds are independent of the specific disk-scheduling algorithm employed, the number of clients in the system, the video bit-rate, the requests arrival pattern, and even the number of servers in the system. Unlike many existing push-based designs, our results show that the buffer requirement under the client-pull service model is invariant to the system scale (i.e., number of servers in the system). Hence, the scalability of the studied pull-based parallel video server architecture will not be limited by the cost of server buffers.

Acknowledgment

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

References

[1] 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, pp. 1173–1183, 1996.

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

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

[4] R. Buck, "The oracle media server for nCube massively parallel systems," in *Proc. 8th Int. Parallel Processing Symp.*, 1994, pp. 670–673.

[5] H. Taylor, D. Chin, and S. Knight, "The magic video-on-demand server and real-time simulation system," *IEEE Parallel Distrib. Technology: Syst. and Applic.*, vol. 3, no. 2, pp. 40–51, 1995.

[6] C. Bernhardt and E. Biersack, "The Server Array: A Scalable Video Server Architecture," in *High-Speed Networks for Multimedia Applications*. Norwell, MA: Kluwer, 1996.

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

[8] W. J. Bolosky, J. S. Barrera, 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. 6th NOSSDAV*, Zushi, Japan, Apr. 1996, pp. 97–104.

[9] M. M. Buddhikot and G. M. Parulkar, "Efficient data layout, scheduling and playout control in MARS," in *Proc. NOSSDAV'95*, Durham, NH, Apr. 1995, pp. 318–329.

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

[11] Y. B. Lee and P. C. Wong, "A server array approach for video-on-demand service on local area networks," in *Proc. IEEE INFOCOM'96*, San Francisco, CA, Mar. 1996, pp. 27–34.

[12] J. Y. B. Lee, "Concurrent push—A scheduling algorithm for push-based parallel video servers," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 467–477, Apr. 1999.

[13] J. Y. B. Lee and P. C. Wong, "Performance analysis of a pull-based parallel video server," *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, pp. 217–231, Dec. 2000.

[14] 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*, Keele, U.K., Apr. 5–7, 1994, pp. 140–143.

[15] A. Reddy, "Scheduling and data distribution in a multiprocessor video server," *Proc. 2nd IEEE Int. Conf. Multimedia Computing and Systems*, pp. 256–263, May 1995.

[16] R. Tewari, R. Mukherjee, and D. M. Dias, "Real-Time Issues for Clustered Multimedia Servers," IBM, Res. Rep. RC20020, June 1995.

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

[18] M. Wu and W. Shu, "Scheduling for large-scale parallel video servers," in *Proc. 6th Symp. Frontiers of Massively Parallel Computation*, Oct. 1996, pp. 126–133.

[19] C. S. Wu, G. K. Ma, and B. S. P. Lin, "A scalable architecture for video-on-demand servers," *IEEE Trans. Consumer Electron.*, vol. 42, pp. 1029–1036, 1996.

[20] J. Y. B. Lee, "Parallel video servers—A tutorial," *IEEE Multimedia*, vol. 5, pp. 20–28, June 1998.

[21] J. Y. B. Lee and P. C. Wong, "Design and performance evaluation of a multimedia web server," *J. Vis. Commun. and Image Rep.*, vol. 9, pp. 183–193, Sept. 1998.

[22] A. L. N. Reddy and J. C. Wyllie, "I/O issues in a multimedia system," *IEEE Comput.*, vol. 27, no. 3, pp. 69–74, Mar. 1994.

[23] A. N. Mourad, "Issues in the design of a storage server for video-on-demand," in *Proc. ACM Multimedia Systems*, vol. 4, 1996, pp. 70–86.

Jack Y. B. Lee is an Assistant Professor in the Department of Information Engineering, Chinese University of Hong Kong. His research interests include distributed multimedia systems, fault-tolerant systems, and Internet computing.