A Segmentation-Based Approach to Delivering Multimedia Files in Content Distribution Networks

Mengkun Yang Zongming Fei
Department of Computer Science, Universtiy of Kentucky
Lexington, KY 40506-0046, U.S.A.
Email: {myang0,fei}@cs.uky.edu

***Abstract*— In this paper we propose a novel segmentation-based approach for delivering multimedia files in content distribution networks. Each media file is divided into segments and the number of copies distributed to replica servers for each segment is designed in such a way that we can reduce the storage requirement at the replica servers, accommodate the bursty arrival of client requests, and at the same time achieve a small latency for the clients. We analyze the effect of the size of segments on the total storage requirement and give a lower bound. We design a scheduling algorithm that can take advantage of the increased capacity of access links of clients. Our simulations show that the segmentation-based approach can reduce the latency and the rejection rate significantly over those approaches that deliver the whole or the prefix part of media files to replica servers.**

I. Introduction

Multimedia streaming is one of the key techniques for many multimedia applications such as distance learning, digital libraries, home entertainment. Multicast has been considered as one important technique for efficient delivery of popular multimedia files to clients [1]. While the requests that can be batched together (possibly with the help of patching [2]) can be served by the same multicast group [3], requests coming at different time and/or asking for different files have to be served by individual streams. We find that in these cases the media files are delivered again and again from the origin server to clients. This problem can be solved by using the content distribution networks (CDNs) to store media files at replica servers close to clients.

The problem with using CDNs for multimedia streaming is that streaming files are usually large and it is quite possible that each replica server can no longer store all media files in the same way as we use CDNs for delivering traditional web documents, which are much smaller. *Prefix caching* schemes [4]–[6] recognize the same problem and propose to store only the prefix part of a video at the replica servers rather than the whole file. When a client makes a request, it will get the prefix of a video file from the replica and get the rest part (suffix) from the origin server. The problem with the prefix caching scheme is that the load on the origin server for delivering the suffix can be extremely high and this may have severe implications on the performance.

This work was supported in part by the National Science Foundation under Grants CCR-0204304 and EIA-0101242.

In this paper, we propose a segmentation-based approach for delivering stored media files in CDNs. We make use of the CDN architecture and solve the problems brought up by the application of streaming large media files. Our goal is to minimize the storage requirement at the replica servers, and at the same time reduce the initial latency, guarantee the smooth playout and accommodate the bursty arrival of client requests. We use video as an example throughout this paper, but the approach is applicable to other media files.

A video file is divided into segments, and not only the first segment is delivered to replicas but other segments as well. Clients get the segments of a video file from its attached replica server or other cooperating replica servers. The number of copies distributed to replica servers for each segment is designed to reflect the timing constraints and the frequency of requests for it. For example, the initial segment has to be delivered to clients as soon as possible to reduce the initial delay, while streaming of other segments can be more flexible because we can use techniques such as prefetching. Another issue in the segmentation approach is the size of each segment. The size of segments can be equal, increase linearly or exponentially. We will study the implications of different size series on the storage requirement.

We have seen that the bandwidth of the last hop is constantly improving. It is possible that end users may access multiple streams from the replica servers located at the edges. Our scheduling algorithm can take advantage of this increased capacity and have clients download future segments from a different server while playing out the current segment. Thus the system can better service the client requests.

Segmentation of video files was first proposed in broadcast video-on-demand systems [7], [8]. These schemes also divide video files into segments and each segment is broadcast in one dedicated channel in cycles even if there are no requests for the video. They assume that there are stable, dedicated broadcast/multicast channels from the origin server to all clients. In contrast, our scheme does not depend on multicast, though the performance can be improved if some multicast scheme is available. Segmentation of multimedia files is also related to the data striping technique used in large storage servers [9], [10], where the goal was to improve the disk access speed and reliability.

Segmentation of video files was also discussed in the caching environment [11]. A fixed pattern (called silo) is used

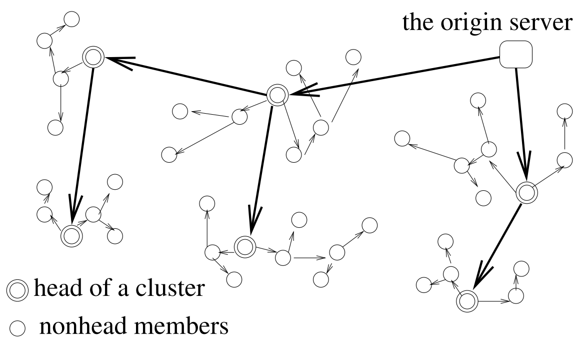

Fig. 1. A two-tier architecture

TABLE I
NOTATIONS USED IN THE PAPER

Symbol	Definition
$\mathcal{L}$	size of a video
$\mathcal{T}$	playout duration of a video
n	number of segments of a video
$w(i)$	weight of the size of segment i
$L(i)$	size of segment i
$f(i)$	number of copies of segment i
λ	arrival rate of requests for a file
ρ	playout rate of a video
k	number of clients a replica can serve
S	total storage requirement for a video

to determine the size of each segment. Probabilities of caching a giving segment are also derived. However, they did not discuss the relationship between the size of segments and the probabilities. In contrast, our main focus is on CDNs and we reveal the relationship between the size and the number of copies for each segment.

The rest of the paper is structured as follows. Section II describes the proposed segmentation-based approach. Section III discusses the implementation issues. Performance results are presented in Section IV. We conclude the paper in Section V.

II. A SEGMENTATION-BASED APPROACH

A. A Two-Tier Model for CDN

We assume a two-tier model for organizing replica servers in the CDN. The replica servers are no longer in a flat structure but divided into groups. The basic principle is that the replica servers in a group should be relatively close in the network distance and they will cooperate to serve the content to clients attached to any replica server in the group. Each group is called a cluster and there is a head for each cluster. It is possible that an overlay distribution network is established among the cluster heads and the origin server. The distribution within a cluster can use an internal overlay network or simple unicast deliveries. Fig. 1 shows an example of two-tier model of CDN where the heads of clusters are connected by an overlay network, which is further extended from the head of each cluster to its non-head members.

Each media file is divided into segments and each server cluster will have a complete copy of the video, i.e., all the segments. This will guarantee that any video can be served only from the replica servers in the same cluster of the replica server attached by the client. However, it is possible that a segment is stored in multiple servers in a cluster to increase the capability of the cluster to serve this segment. In this case, we need to solve the server selection problem because we have to decide which replica server to use.

An alternative to the segmentation approach is that we can let different replica servers store different video files and let them collaborate to serve clients. It is called *unsegmented* approach in the rest of this paper. Assuming that a video file is stored at a specific server, we find that the number of requests we can handle is limited by the capacity of that server. We adopt the segmentation approach, in which segments of a file are distributed to multiple servers in the cluster. The number of requests we can handle for this file is increased because the load is distributed to all the servers having segments of this file. Therefore, it provides a higher capacity to serve a specific file with the same storage requirement at a given time [1]. The cluster can thus handle the bursty arrival of requests for this file better.

B. Notations

In this section, we introduce several notations used in the analysis. Consider a video that is divided into n segments. Usually the number of replica servers in a cluster is larger than n, and we can distribute different segments of a file to different replica servers. The length of each segment and the number of copies of each segment in a cluster are two primary design parameters. The length of each segment is defined by a weight function w and the length $L(i)$ of segment i is proportional to $w(i)$. Therefore, we have $L(i) = \frac{w(i)}{\sum_{j=1}^{n} w(j)} * \mathcal{L}$, where $\mathcal{L}$ is the total size of the video.

The number of copies of each segment is given by a function f. The number of copies of segment i is $f(i)$. Therefore the total storage required for a video in the cluster is $S = \sum_{i=1}^{n} f(i)L(i)$. The goal is to minimize the storage requirement in a cluster of replica servers and at the same time to be able to accommodate the bursty arrival of client requests.

We assume the arrival rate of requests for a video in a cluster is λ (requests per second) and the playout rate of the video is ρ (megabits per second). Replicas may have different capacities and they may serve different numbers of clients at the same time. To simplify our analysis we assume that each replica can serve k clients at the same time, i.e., it can send out k streams at the playout rate of a video. Table I summarizes the notations used in the analysis.

C. Lower Bound on Storage Requirement

We first consider an ideal case that gives the minimal storage requirement (a lower bound) for a specific file, under a given

[1]We do recognize that the total capacity to serve all files cannot be increased with a unicast-only approach we use, but the segmentation approach provides the flexibility of contributing resources from multiple servers in a cluster to serve a specific file at a given time.

arrival rate and the assumption that a video is divided into n segments. Equal segmentation is the simplest one, i.e., $w(i) = w(1)$ for $1 \leq i \leq n$. The size of each segment is $L(i) = \mathcal{L}/n$ and the duration for streaming one segment is $t_s = \mathcal{L}/(n\rho)$. In this ideal case, requests come evenly every t_s second. Since the duration of a video is $\mathcal{T} = \mathcal{L}/\rho$, the total number of requests during a video playout is $\lambda\mathcal{T} = \lambda\mathcal{L}/\rho$. The maximal possible number of requests in one segment time is $\lceil \lambda\mathcal{L}/(\rho n) \rceil$. Considering that a replica server can serve k requests at the same time, we will be able to meet the request requirement by setting $f(i) = f(1) = \lceil \lambda\mathcal{L}/(\rho k n) \rceil$. The total storage requirement is $S = \sum_{i=1}^{n} L(i) f(i) = \sum_{i=1}^{n} \frac{\mathcal{L}}{n} * \lceil \frac{\lambda\mathcal{L}}{\rho k n} \rceil = \mathcal{L} \lceil \frac{\lambda\mathcal{L}}{\rho k n} \rceil$

For comparison, we also derive the storage requirement in the *unsegmented* case where the whole video will be stored in a single replica. To satisfy $\lambda\mathcal{L}/\rho$ requests during the video playout time, we need to have $\lceil \lambda\mathcal{L}/(\rho k) \rceil$ copies of the video, where k is the number of requests a replica server can serve at the same time. The storage requirement is $S = \mathcal{L} \lceil \frac{\lambda\mathcal{L}}{\rho k} \rceil$. To serve the same number of requests for a specific video, the *unsegmented* approach will need up to n times storage as the segmentation approach in the ideal case discussed above.

D. Storage Requirement with Bursty Traffic

We consider a more general case in this section. In practice, requests for a given video can come at any time. For example, the average number of requests during a video playout time is $\lambda\mathcal{L}/\rho$, and it is possible that all the requests arrive within one segment time. The arrival can be more bursty than this case. However, with the limited resource in a cluster we cannot satisfy all arbitrary arrival patterns. Our assumption is that the arrival during a segment time is at most n times the average rate, or equal to the average number of requests that may arrive during the whole video playout time. We analyze the storage requirement in this case and design the series functions w and f that determine the size and the number of copies for each segment, respectively.

Our approach is to leave w open and find out the number of copies for each segment in order to accommodate the above-mentioned burstiness of the traffic. The worst case scenario is that all requests within a video playout time come during the first segment time. The total number of these requests is $\lambda\mathcal{L}/\rho$. To satisfy these requests, the number of copies for the first segment $f(1) = \lceil \lambda\mathcal{L}/(\rho k) \rceil$, considering that a replica can serve k requests at the same time.

While the first segment must be served during the first segment time, the second segment has more flexibility. It can be either prefetched by the client during the first segment time, or served by a replica server during the second segment time. Therefore, the time we can serve the second segment is the total time of playing out the first segment and the second segment. As long as the second segment can be streamed to clients within the second segment time, the continuous playout requirement is satisfied. We assume that different segments from the same video will be placed on different replica servers. Under this condition, the number of clients that can be served by one copy of the second segment is $k * \lfloor \frac{L(1)+L(2)}{L(2)} \rfloor = k * \lfloor \frac{w(1)+w(2)}{w(2)} \rfloor$. The total number of copies for the second segment is: $f(2) = \lceil \frac{\lambda\mathcal{L}/\rho}{k * \lfloor \frac{w(1)+w(2)}{w(2)} \rfloor} \rceil$.

In general the i-th segment can be served during the first, second, until i-th segment. Therefore, one copy of i-th segment can serve $k * \lfloor \frac{\sum_{j=1}^{i} L(j)}{L(i)} \rfloor = k * \lfloor \frac{\sum_{j=1}^{i} w(j)}{w(i)} \rfloor$ client requests. The total number of copies of the i-th segment is: $f(i) = \lceil \frac{\lambda\mathcal{L}/\rho}{k * \lfloor \frac{\sum_{j=1}^{i} w(j)}{w(i)} \rfloor} \rceil$.

So the total storage required is

$$S = \sum_{i=1}^{n} f(i) * L(i) = \sum_{i=1}^{n} \lceil \frac{\lambda\mathcal{L}/\rho}{k * \lfloor \frac{\sum_{j=1}^{i} w(j)}{w(i)} \rfloor} \rceil * \frac{w(i)}{\sum_{j=1}^{n} w(j)} \mathcal{L}$$

The determining factor then is the series function w. We first consider a simple case where all the segments are of equal size, i.e., $w(i) = 1$ for all i. We will use EQ to denote this series function in the figures. We can get $L(i) = \mathcal{L}/n$ and $f(i) = \lceil \frac{\lambda\mathcal{L}/\rho}{k * i} \rceil$. So the total storage requirement is $S = \sum_{i=1}^{n} \lceil \frac{\lambda\mathcal{L}/\rho}{k * i} \rceil * \mathcal{L}/n$.

We further study the cases in which segments are of unequal size. Here is a list of series functions for w.

1) The size of segments increases exponentially.
   - EXP1: The simplest series is 1, 2, 4, 8, etc. That is, $w(i) = 2^{i-1}$.
   - EXP2: The size of segment i is the sum of all segments up to $i-1$, i.e., 1, 1, 2, 4, 8, etc. That is, $w(i) = \sum_{j=1}^{i-1} w(j)$ or in a closed form $w(i) = 2^{max(0,i-2)}$. Then S can be simplified as: $S = \sum_{i=1}^{n} \lceil \frac{\lambda\mathcal{L}/\rho}{k * 2} \rceil * 2^{\frac{max(0,i-2)}{n-1}} * \mathcal{L}$
   - EXP3: The series is 1, 2, 2, 4, 4, 8, 8, etc. That is, $$w(i) = \begin{cases} 1 & \text{if } i = 1 \\ 2 & \text{if } i = 2 \\ w(i-1) & \text{if } w(i-1) \neq w(i-2) \\ w(i-1) * 2 & \text{if } w(i-1) = w(i-2) \end{cases}$$
2) The size of segments increases linearly.
   - LIN1: The size of segment i is proportional to i, i.e., $w(i) = i$.
   - LIN2: The series function is 1,1,2,2,3,3,4,4, ..., i.e., $w(i) = \lceil i/2 \rceil$. We can get $f(i) = \lceil \frac{\lambda\mathcal{L}/\rho}{k * (\lfloor i/2 \rfloor + 1)} \rceil$ and $S = \lceil \frac{\lambda\mathcal{L}/\rho}{k * (\lfloor i/2 \rfloor + 1)} \rceil * \frac{w(i)}{\sum_{j=1}^{n} w(j)} \mathcal{L}$.

In Fig. 2, we analyze the storage requirement of these different series functions. Also included are the *unsegmented* case and the ideal case in the previous subsection, We assume the video length equals 120 minutes, the playout rate is 1.5Mbps, and the request rate is 5/minute. Though the ideal case usually will not be able to meet the requirement of client requests, it provides a comparison baseline. We can find that the simple equal segmentation (EQ) requires much less storage than the *unsegmented* approach. For example, if the number of segments is 35, we need only one eighth of storage required by the unsegmented approach. The storage requirement decreases

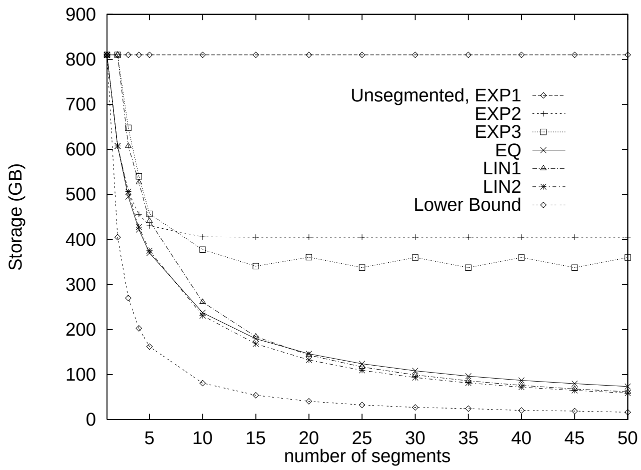

Fig. 2. Size of Segments

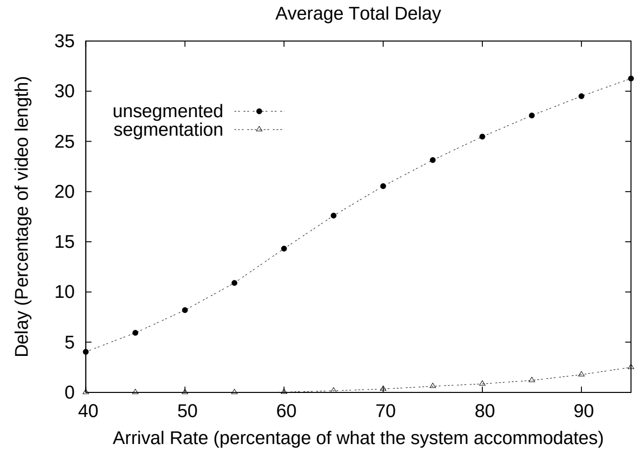

Fig. 3. Segmentation vs. Unsegmented

as the number of segments increases. All exponential series need much larger storage than the two linear series, which are very close to the equal size series. Because of the simplicity of the equal size series, we will use it in the performance evaluation unless otherwise noted.

III. Implementation Issues

When a media file is delivered to the head of a cluster of replica servers, it will divide the file into segments. The size of each segment is based on the series function w. These segments will be distributed to different servers in the cluster. Initially only one copy will be maintained for each segment. As the request rate to a given file increases, the cluster may make a decision to increase the number of copies for each segment, based on the function f. The information about the locations of all segments will be distributed to all replica servers in the cluster.

When a replica receives a request from a client, it will act as the coordinator for this request. We have developed a communication protocol for this coordinator to exchange information with other replica servers to find out the available time slots in which they can stream the segments asked by this request [12]. It is possible that it gets replies from multiple replicas having a specific segment. We designed a scheduling algorithm for the coordinator to determine the appropriate streaming time for each segment of the file requested by the client.

Assume that the client makes the request at time t_{rqst}. Since the length of segment j is $L(j)$, the playout time of segment j is $s(j) = L(j)/\rho$. Ideally, there will be no delay if the client can start downloading segment j $(0 \leq j \leq N-1)$ at time $t_j = t_{rqst} + \sum_{k=0}^{j-1} s(k)$. However we may have to download segment j before or after t_j because the time slots starting at time t_j on all channels of the replica server may have already been reserved for other requests. While an actual downloading time before t_j will not cause any problem, a downloading time after t_j will cause the client to start playing out the whole media file after some initial latency. Specifically, if the scheduled downloading time is t'_j for segment j, the latency caused by it is $\Delta t_j = max(0, t'_j - t_j)$. The final overall initial latency for this client is the maximum of the latencies caused by all segments, i.e., $\max_{0 \leq j \leq N-1} \Delta t_j$. The scheduling algorithm will try to minimize this latency.

Our algorithm can deal with the cases where multiple servers have the same segment requested by the client. Also clients may have different capacities that determine the number of segments it can download at the same time. The scheduling algorithm can make the decision based on these input parameters and optimize the performance for clients. The clients may be instructed to play out current segment and prefetch several future segments at the same time. The details of the algorithm can be found in [12].

IV. Performance Evaluation

We conduct simulations to evaluate the performance of the segmentation-based approach for multimedia streaming. In the simulation we have 200 replica servers and 5000 clients. Every replica has up to 20 channels to transmit media file segments, and a client can download from 4 channels simultaneously. The system provides on-demand service for 100 media files and each file is 3600 second long. Client requests are modeled as a Poisson arrival process and the frequency for requesting different files are approximated by a Zipf-like distribution with parameter $\alpha = 0.733$ [13]. The theoretical maximum request rate that the system can possibly support is $20 \times 200/3600$, which is the case when *all* servers' channels are kept *always* busy. The actual arrival rate in the simulation is set as a percentage of this rate.

In the segmentation scheme, each video is equally divided into 15 segments and the number of copies for each segment is calculated based on the segment size series, as well as the request arrival rate for the file it belongs to. The performance measures we will show here is the latency of clients.

In Fig. 3 we compare the segmentation-based approach (*segmentation*) with the *unsegmented* approach. The total storage at replica servers is the same, but the segmentation-

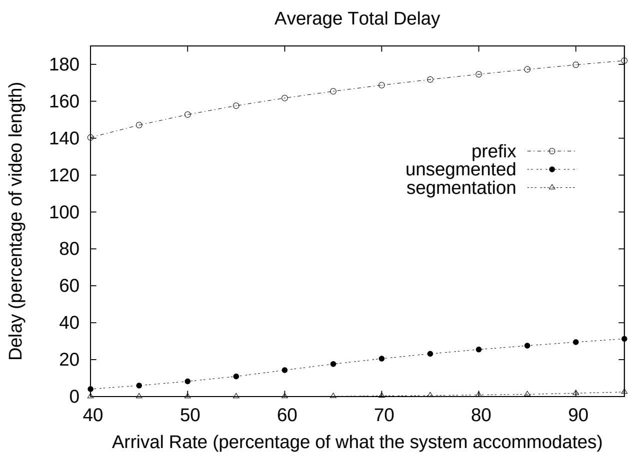

Fig. 4. Different approaches

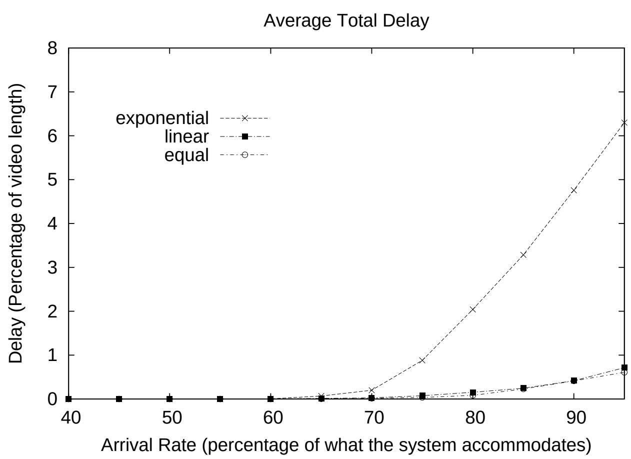

Fig. 5. Different Segmentation Series

based approach results in much smaller delay. For example, when the arrival rate is 80%, the delay of the segmentation-based approach is less than 1% of video length while the *unsegmented* approach is about 25% of the video length. Fig. 4 includes the *prefix* caching approach. We set every prefix 720 seconds long, i.e., 20% of the file length. The suffix of every file has only one copy in the system, while the prefix is replicated as many as the system storage permits. We can see the average delay for the prefix caching is more than one whole video length longer than the other two approaches. This is because the extremely high load on the server storing suffix parts results in a long waiting time for client requests.

In the next experiment we studied the effect of three segmentation series: (1) equal-sized series (EQ); (2) linear series (LIN2); (3) exponential series (EXP2). In this experiment, the number of segments in a file and the number of client downloading channels are both set to 8. Fig. 5 depicts delays in these cases. The "exponential" series has a much larger delay than the "equal-sized" and "linear" series because the size of later segments in the "exponential" series is so large that they use most of the system resources. Therefore the benefits of the segmentation are a bit compromised. This result is consistent with the analysis in Section II.

Another performance measure we studied is the *rejection rate*, which is the percentage of clients rejected because their delay is larger than a threshold value. The simulation shows that the segmented approach reduces the rejection rate significantly over the unsegmented and the prefix approaches. We also studied the effects of the number of downloading channels of clients. We found that even when the number of downloading channels of clients is one or two, the latency and the rejection rate are still much lower than the unsegmented and the prefix approaches, though the performance can be further improved if clients have more downloading channels. Unfortunately we cannot show the results due to the space limitation.

V. Concluding Remarks

In this paper we presented a novel segmentation-based approach for streaming multimedia files over the Internet by using content distribution networks. We analyzed the storage requirement of various segmentation series. Our simulations demonstrate a significant performance gain of the proposed approach.

References

[1] J. M. Almeida, D. L. Eager, M. Ferris, and M. K. Vernon, "Provisioning content distribution networks for streaming media," in *Proceedings of IEEE Infocom'02*, June 2002, pp. 1746–1755, New York, NY.

[2] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proceedings of ACM Multimedia'98*, 1998.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "On optimal batching policies for video-on-demand storage servers," in *Proceedings of IEEE Int'l Conference on Multimedia Systems'96*, 1996.

[4] S. Ramesh, I. Rhee, and K. Guo, "Multicast with cache (mcache): an adaptive zero-delay video-on-demand service," in *Proceedings of IEEE Infocom'01*, 2001.

[5] Z.-L. Zhang, Y. Wang, D. H. Du, and D. Su, "Video staging: A proxy-server-based approach to end-to-end video delivery over wide-area networks," *IEEE/ACM Transactions on Networking*, vol. 4, no. 8, pp. 429–442, August 2000.

[6] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE Infocom'99*, April 1999.

[7] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 3, no. 4, pp. 197–208, May 1996.

[8] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM Sigcomm'97*, 1997.

[9] J. R. Santos and R. Muntz, "Comparing random data allocation and data striping in multimedia servers," in *Proceedings of ACM Sigmetrics*, June 2000, Santa Clara, CA.

[10] P. Shenoy and H. M. Vin, "Efficient striping techniques for variable bit rate continuous media file servers," *Performance Evaluation*, vol. 38, no. 3, pp. 175–199, December 1999.

[11] Y. Chae, K. Guo, M. M. Buddhikot, S. Suri, and E. W. Zegura, "Silo, rainbow, and caching token: Schemes for scalable, fault tolerant stream caching," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1328–1344, September 2002.

[12] M. Yang and Z. Fei, "A fine-grained peer sharing technique for delivering large media files over the internet," in *Proceedings of Eighth International Workshop on Web Content Caching and Distribution (WCW'03)*, September 2003, Hawthorne, NY.

[13] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence, and implications," in *Proceedings of INFOCOM'99*, March 1999, pp. 126 –134, New York, NY.