Distributed Servers Architecture for Networked Video Services

S.-H. Gary Chan, *Member, IEEE*, and Fouad Tobagi, *Fellow, IEEE*

***Abstract*—In an on-demand video system, the video repository generally has limited streaming capacities and may be far from the users. In order to achieve higher user capacity and lower network transmission cost, distributed servers architecture can be used, in which multiple local servers are placed close to user pools and, according to their local demands, dynamically cache the contents streamed from the repository. We study in this paper a number of caching schemes as applied in the local servers depending on whether the repository is able to multicast movie contents to the local servers or not, and whether the local servers can exchange their cached contents among themselves or not. Our caching schemes keep a circular buffer of data for the movie requested, and hence movies are partially cached. By adjusting the size of the buffer, such caching is able to achieve better tradeoff between network channels and local storage as compared to the traditional caching in which a movie is treated as an entity. For each caching scheme, we study the tradeoff between the local storage and the network channels, and address how the total cost of the system can be minimized by appropriately sizing the buffer. As compared to a number of traditional operations (request batching and multicasting, true-VOD, etc.), we show that distributed servers architecture is able to achieve much lower system cost to offer on-demand video services.**

***Index Terms*—Caching schemes, distributed servers architecture, network channels and local storage tradeoff, unicast and multicast, video-on-demand.**

I. Introduction

ADVANCES in computer and networking technologies have made on-demand video services (video-on-demand, or VOD) a reality. There are many important video services pertaining to entertainment, education, advertising and information, such as movie-on-demand, distance learning, home shopping, and interactive news [1]–[4].

In an on-demand video system (i.e., videos are displayed upon user request with negligible delay), a number of repository servers such as tertiary libraries or jukeboxes (collectively referred to as a repository) store all the video contents of interest to a large number of geographically distributed users. If videos were to be streamed directly to the users, the user capacity in the system would be limited by the streaming capacity of the repository. Such capacity can be increased by using a hierarchy of servers, in which multiple streaming servers cache the movies delivered from the repository and stream them to the users.[1] If the streaming servers were co-located with the repository, the transmission cost incurred in streaming videos to remote users might be high. To overcome this problem, as well as to take advantage of the access locality and demand characteristics of the user pool, the streaming servers may be placed close to the user clusters, thus forming a "distributed servers architecture." Such a distributed servers architecture in fact has been discussed previously [3], [5]. The system is able to achieve scalable storage and streaming capacities by introducing more repository servers and local servers as the traffic increases.

We consider in this paper that a cost is associated with storing a movie in a local server depending on how much and for how long the storage is used (such a "storage utility" model is in fact being offered by some companies, in which one gets storage whenever and as much as one needs, and pays only for what one uses). Suppose that a 1-GB disk costing today about \$200 is in use for a certain number of hours per day, say, six hours. The disk is to be amortized over a period of one year. The cost of storage is then given by \$200/(365 days $\times$ 6 h/day) $=$ \$0.091/h per GB. Consider the streaming rate of a movie to be $b_0 = 5$ Mb/s (MPEG-II quality). Then one minute of video data (of 0.0375 GB) costs 0.091/h $\times$ 0.0375 $=$ \$3.42 $\times 10^{-3}$/h $=$ \$5.71 $\times 10^{-5}$/min of storage. From above, we see that the storage cost depends not only on how large the data is, but also on how long it is stored. If we take into consideration the cost due to data redundancy for fault tolerance (e.g., by means of mirroring or introducing error checking overheads) and the cost of server streams (which may be a function of the storage used), the storage cost would be higher.

Furthermore, we consider that a cost is associated with streaming a movie from the repository to the remote users. The cost of the network channel may range from low (e.g., when the Internet is used) to quite high (e.g., when satellite channels are used). In Table I, we show some values of the channel cost per minute, β (for the case $\beta =$ \$0.03/(min·channel), the delivery of a 100-min movie costs \$3, a reasonable cost for on-demand services). Also shown is a parameter γ given by the ratio of the storage cost given above (i.e., \$5.71 $\times 10^{-5}$/min of storage for a 1-min video clip) to β (γ is hence the relative cost of storage with respect to that of the network channel). From the table, we see that γ likely ranges from 10^{-4}–10^{-2} (in channel/min, or simply, in /min) for on-demand services.

Manuscript received January 21, 2000; revised July 20, 2000; approved by IEEE/ACM Transactions on Networking Editor E. Biersack. This work was supported in part by the Center for Telecommunications at Stanford University, and by the Areas of Excellence (AoE) on Information Technology funded by the University Grant Council in Hong Kong, and by a Direct Allocation Grant of Hong Kong Research Grant Council. This work was presented in part at the IEEE International Conference on Communications, June 1999.

S.-H. G. Chan is with the Department of Computer Science, Hong Kong University of Science and Technology, Kowloon, Hong Kong (e-mail: gchan@cs.ust.hk).

F. Tobagi is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 1063-6692(01)03234-4.

[1]In this paper, we use the terms "movie" and "video" interchangeably to refer to a file of long streaming duration, say, more than 30 minutes.

TABLE I
VALUES OF β AND γ (BASED ON 1 GB DISK OF \$200, AMORTIZED OVER A YEAR WITH USAGE OF 6 H/DAY, AND $b_0 = 5$ Mb/s)

β (\$/(channel·min))	0.3	0.03	0.003
γ (/min)	2×10^{-4}	2×10^{-3}	2×10^{-2}

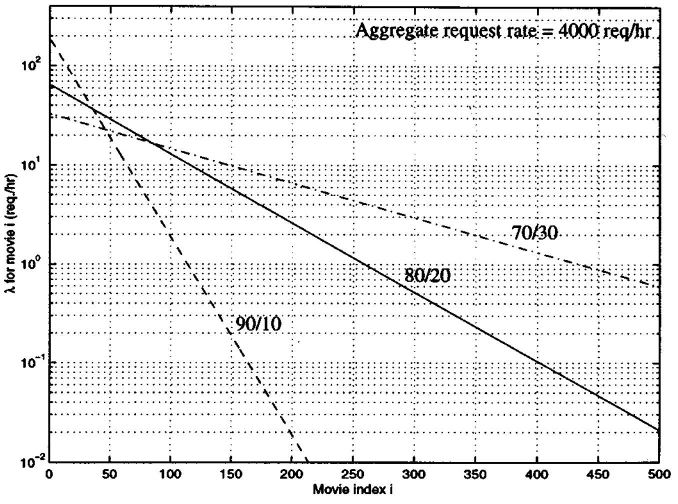


Fig. 1. λ for movie i in a video system with 500 movies and $\Lambda = 4000$ req/h (geometric video popularity).

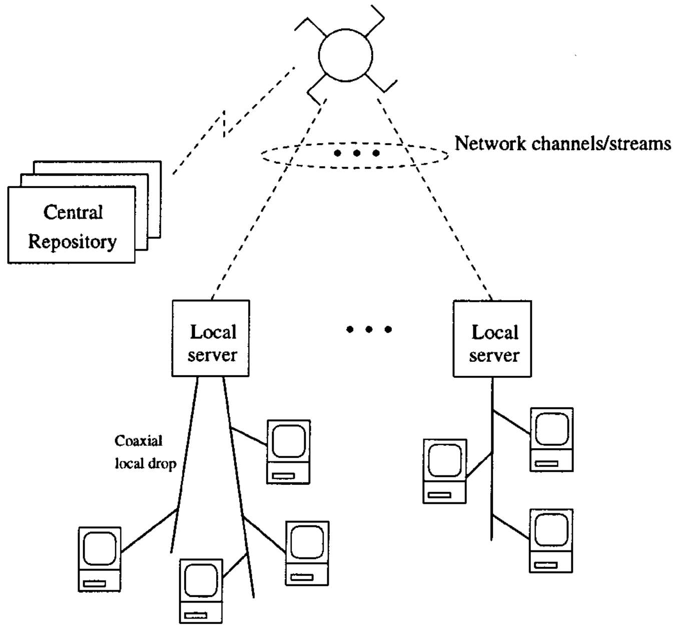


Fig. 2. Distributed servers architecture for networked video services.

Consequently, there is a tradeoff between using a long-haul network channel to deliver a movie and storing the movie locally (thus saving on communication cost): if the demand for the movie is high, we should store the movie locally so as not to incur too often network transmission cost; on the other hand, if the demand is low, we should stream the movie directly from the repository (the so-called "true-VOD" case) so as not to incur too much local storage cost. For the movies of intermediate popularity, we should cache them *partially* in order to achieve the tradeoff.

Since some movies are popular while others are not, and the popularity may change over time, there is a need to decide which movie and how much of the movie should be cached locally given its request rate in order to minimize the system cost. Given the dynamic nature of the request rate, it is important that such a decision be made continuously over time. (In other words, the repository and local servers may have to be constantly exchanging movies.) For instance, the introduction of a new lecture or movie title can change the popularity of some movies, thereby making some no longer worth storing locally. As an example, consider a geometric video popularity model in which the access probability of the movies follows a geometric distribution (i.e., the ratio of the request rate of movie i to movie $i-1$ is a constant less than or equal to 1 [6], [7]). We show in Fig. 1 the request rate for movie i in a system of 500 titles with the aggregate request rate $\Lambda = 4000$ req/h given different popularity skewness indicated by r/m (the notation r/m represents that r% of the requests ask for m% of the movies). We see that some movies are very popular, with an average of tens to hundreds of concurrent users (assuming a movie length of, say, 90 minutes), while some others are not popular at all, with an average of less than one concurrent user. As the skewness increases, the difference in popularity between titles also increases. The request rate of a movie can differ quite markedly by orders of magnitudes.

In a distributed servers architecture, the local servers may not be able to exchange information among themselves due to, for example, limited processing capacity (for frequent content exchanges and updates), network limitation (e.g., disjoint networks or limited network capacity), or a lack of incentive to do so (e.g., uncooperative service providers or security issues). Each of the local servers therefore obtains its movies only from the repository and operates independently from each other, and the repository has to unicast the movies to the local servers. This is shown in Fig. 2 for a cable-TV system, in which the head-end servers serve their local communities through coaxial local drops while the central repository unicasts the movies with the use of network channels such as the Internet or satellite channels.

On the other hand, a service provider may set up a number of local servers, each serving a region. If multicast network channels are available, they can be used to deliver videos to the local servers to decrease the network channel requirement and cost. A local server can get its data by joining any existing multicast groups.

To further decrease the long-haul transmission cost, the local servers, instead of always obtaining the data from the repository, can be connected by a network so that they can communicate and exchange their data with each other. This is possible for servers co-located in, for example, a campus network, an entertainment network with cooperative service providers, or a private enterprise network. In general, the transmission cost between the local servers are low relative to that from the repository. By buffering part of the streamed data, the servers can serve the subsequent local request(s) among themselves, hence conserving bandwidth from the repository to the server network.

The repository may multicast movies to the local servers, or it may unicast the movie to a particular local server, which in turn unicasts/multicasts the movie to the other servers using the network between the local servers.

In this paper, we study a number of caching schemes as used in the local servers. All schemes employ a circular buffer so that data is stored in a local server for a certain maximum amount of time, hence keeping only a portion of the movie locally (i.e., *partial* caching). In other words, the schemes keep a window along the movie playtime, so that all requests arriving within the window are served directly from the local cache (known as a *cache group*). (Keeping a window of the movie in the local server also offers some limited degree of interactivity to the users.) Clearly, the larger is the window/buffer, the larger is the storage requirement in the local server and the lower is the network channel cost. Therefore, by adjusting the size of the window, the network channel and local storage can be effectively traded off with each other. We consider a network with the channels properly sized so that channels can be acquired on demand (i.e., the probability of running out of channels is low and can be ignored). We also consider that the latency in the central repository is low and can be ignored.

In a distributed servers architecture, the service provider may pay for the costs of the transmission from the repository and the local storage. In this case, it is important to examine the conditions under which a movie should be stored so as to minimize the total cost, given its request rate and the cost functions of storage and network channels. We show that for unicast delivery, if network channel cost is linearly dependent on its holding time, movies should be either entirely stored or not at all and we give the cutoff point in the request rate for this. On the other hand, for multicast delivery or communicating servers, the optimal strategy is to use a circular buffer to cache the movie partially. Multicast is able to lower the system cost compared with the unicast case for a certain range of arrival rate. If servers can communicate with each other, the system cost can be greatly reduced. We show that, by taking advantage of the current low storage cost, the distributed servers architecture is able to achieve much lower system cost (by an order of magnitude or so) than a system based on request batching and multicasting, while offering on-demand services.

This paper is organized as follows. After a brief review of the previous work in Section II, we describe in Section III the caching schemes studied in this paper. In Section IV, we analyze the schemes in terms of their storage and channel requirements, and how their joint costs can be minimized. In Section V we provide illustrative numerical results and comparisons, and in Section VI we compare the cost advantage of a distributed servers architecture with respect to a number of traditional video systems. We conclude in Section VII.

II. Previous Work

We briefly discuss previous work as follows. Barnett and Anido [5] compare the *setup* cost of a centralized video system with a distributed one. Given certain cost functions in storage and streaming, they show that an optimally designed distributed system may achieve lower cost than a centralized one. We differ by studying the *ongoing running* cost of a video system. Schaffa and Nussbaumer study a distributed video system in which servers are configured as a hierarchical balanced tree, and all video files are replicated at a certain level in the tree [8]. While the paper studies the symmetric case in which request rates across the servers at a given tree level are the same, we make no such assumption here. Lie *et al.* [9] and Little *et al.* [10] consider a video system in which video files can be replicated in order to reduce user loss rate. All the work above treat a video file as a single entity, and hence a movie is either completely stored in a local server or not at all. We consider here partial caching, which achieves better tradeoff in storage and bandwidth.

Papadimitriou *et al.* [11] consider a reservation system in which a central repository delivers videos through a series of caching nodes. Given a prespecified viewing schedule, the scheduler determines when, where, and how long a file should be cached in a storage node so as to minimize the total cost. Our work differs in that we consider *on-demand* services. Wang *et al.* [12] consider movie transmission by storing the "bursty" portion of a movie in a local server so as to reduce the requirement in the network bandwidth (hence achieving tradeoff in local storage and network bandwidth). Our work does not assume *a priori* knowledge of the bandwidth profile of a video as in the work. Dan *et al.* [13] consider load-balancing issues among multiple disks in a server. A movie stored in a disk is divided into a number of segments. Depending on the load of a disk, the segment can be dynamically replicated to one of the other disks to increase the throughput. Though partial movie storage is considered, network bandwidth issues (i.e., its requirement and tradeoff with the storage) have not been addressed.

We differ from all the above work in that we study using multicasting and server caching to deliver video data to the local servers, and consider a number of ways to cache the multicast streams. The use of caching in multicast network has been previously studied in the context of "client buffering," in which a user buffers video streams so as to decrease the network transmission cost [14]–[22]. We study here "server caching," in which the server caches data on the client's behalf (and hence a client does not need any buffering). Such a technique leads to storage sharing (and hence a decrease in the overall storage cost), and some new operational schemes (such as the prestoring and precaching schemes in this paper). Server caching was previously studied to offer user interactive capabilities [23]; we differ in studying distributed servers architecture for on-demand services, and examine analytically the tradeoff between storage and network channels.

III. Caching Schemes

In this section, we describe the caching schemes studied. We begin by describing a scheme in which the repository unicasts video contents to each local server. Then we describe the schemes for multicasting. Finally, we describe a scheme for communicating servers. Since storage and network channels can be acquired on-demand, the servicing of requests pertaining to a given movie is independent of the servicing of requests

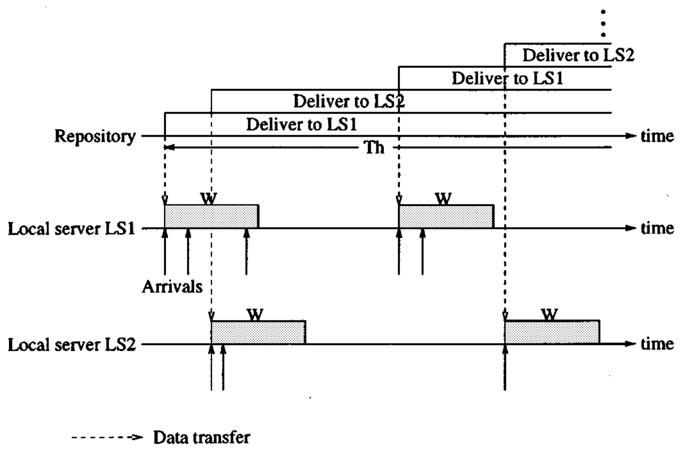


Fig. 3. Scheme for unicast delivery.

for other movies; therefore we can focus our discussion on a given movie. We assume a sufficiently stable networks, and fault recovery and service interruptions in the process of video transmission are not the issues (and hence are not discussed).

A. Unicast Delivery

Since the local servers are operated independently in this case, we can focus on an arbitrary local server with a fixed buffer for each movie. To describe the scheme, let us consider that initially no user is being served in a local server and no movie is cached locally. (For illustration, please refer to the timing diagram in Fig. 3, in which we show a repository and two local servers LS_1 and LS_2.) An arrival at the server leads to an allocation of a network channel of duration T_h minutes to stream the movie from the repository to the local server, while the local server caches the data with a circular buffer of size W minutes (the lightly shaded boxes in the figure). The window size W is hence the data lifetime in the local server ($0 \leq W \leq \infty$). The beginning of the movie is replaced after W minutes, and hence all requests (including the first one) arriving within the window form a cache group (i.e., they are served by the local server with the allocation of a single network channel). Arrivals more than W minutes from the first user in the cache group start a new stream. Clearly, the number of streams needed in the repository at a time is the sum of all the opened (active) channels at that time.

The buffer requirement can be further reduced if, after we have collected a cache group, we "trim the buffer" (by cutting W) and keep only the amount of buffer enough to serve the first and the last arrival in the group. This case has been examined in [24], and it is shown that the saving is not very significant and hence will not be discussed in this paper. (Another way is to dynamically adjust the window size according to the arrival pattern. It has also been shown that such a scheme performs similarly with the case discussed here [24].)

B. Multicast Delivery

We consider the following two schemes, depending on whether a certain portion of the movies are permanently stored in the local servers or not.

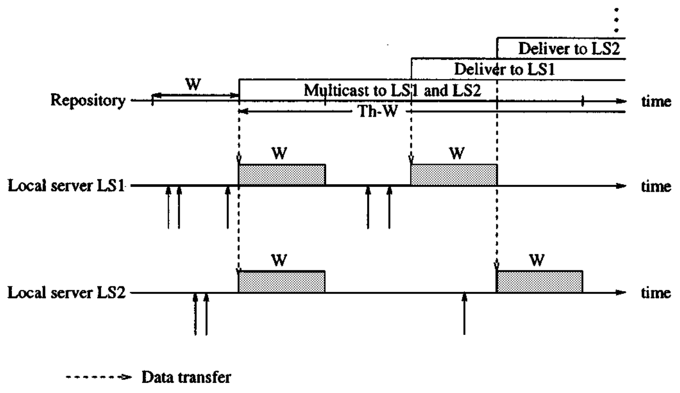
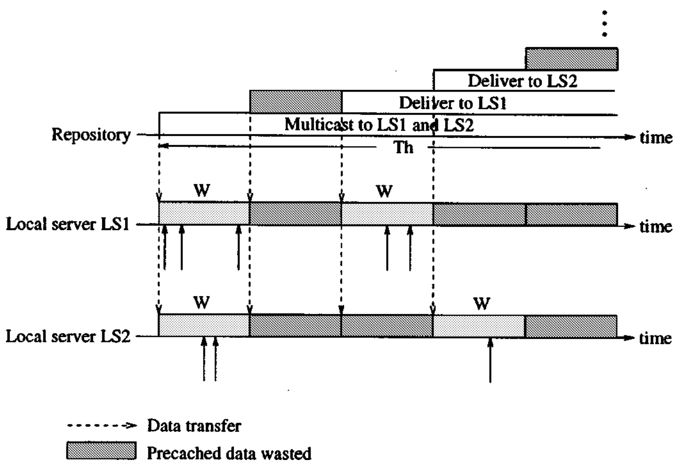


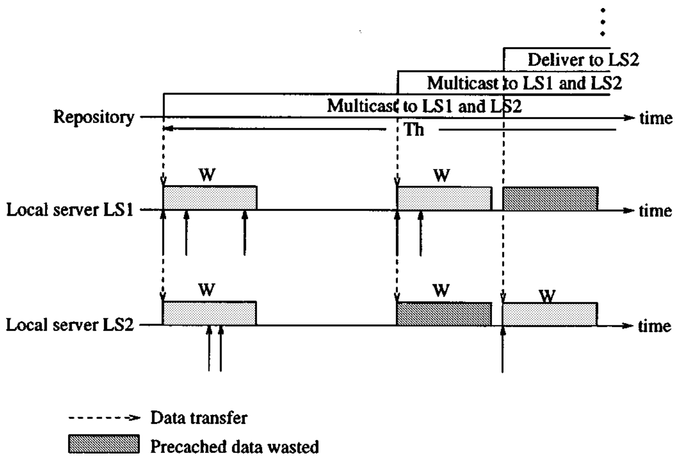
Fig. 4. Prestoring scheme for multicast delivery.

- *Prestoring:* A local server stores permanently the leading portion (the "leader" [7]) of each movie. Let the leader size be W minutes. There is a corresponding periodic multicast schedule from the repository for the movie with slot interval W minutes. At the beginning of each slot, the movie is delivered from its W minutes onwards till its end. (For illustration, please refer to Fig. 4 for a system with a repository and two local servers LS_1 and LS_2.) All requests for the movie arriving at a local server within a multicast interval are first served by its leader at their respective local servers. In the slot following the requests, the repository multicasts the remainder of the movie (of duration $T_h - W$ minutes) to all the local servers which have requests in the previous slot. Since the requests are offset in time, the local server has to cache the multicast streams with a buffer of size W minutes (lightly shaded in the figure) so that, after the display of the leader, the cached data can be used to serve the requests. If there is no request arriving into the system within a multicast interval, the multicast of the movie in the following slot is canceled. Clearly, if $W = 0$, we have a true-VOD system (in which the repository serves the requests directly on the demand basis).
- *Precaching:* In precaching, there is no storage permanently laid aside in the local servers; instead, a local server decides in advance if it should cache a multicast movie (i.e., precache) or not. If data is cached, future arrivals can be served from the local cache; however, if there is no arrival within the window, the buffer would be wasted. If data is not cached, the repository has to serve the requests directly on the demand basis with a unicast stream (i.e., the true-VOD case). We consider the following two schemes depending on whether the multicast schedule is preset (the "periodic multicasting with precaching") or request-driven (the "request-driven precaching").

In the *periodic multicasting with precaching*, the repository multicasts a movie from its beginning at regular intervals of W minutes and a local server precaches the movie using a circular buffer of W minutes. (We show in Fig. 5(a) a case in which the local servers LS_1 and LS_2 both precache data.) If there is any request arriving within the first W minutes of the movie multicast, the request is served from the local cache (cases indicated by the lightly shaded



(a)



(b)

Fig. 5. Precaching schemes for multicast delivery. (a) Periodic multicasting with precaching. (b) Request-driven precaching.

boxes in the figure) and the multicast stream will be held for T_h minutes; otherwise, the buffer would be flushed and the multicast stream would be aborted at the end of the W minutes (cases indicated by the darker boxes in the figure).

The *request-driven precaching* is very similar to the above except that movie multicast is initiated on-demand upon the arrival of a request. We illustrate this scheme in Fig. 5(b), in which both LS_1 and LS_2 precaches data for a time of W minutes. Clearly, the start of multicast streams is driven by request arrival. All requests arriving within the precaching window in a local server can be served from the local buffer (cases indicated by the lighter boxes). If there is no arrival within the window, the buffer is purged (indicated by the darker boxes).

C. Communicating Servers

In this case, video data can be transferred from one server to another using the server network. To describe the scheme, let us consider that initially the system is empty without any cached data and requests. (We illustrate the scheme in Fig. 6 for a system with two local servers.) A request arriving at a local server (server LS_1) leads to the unicasting of the movie from the repository to the local server, which buffers the data for a time W minutes so that local requests arriving within the window can

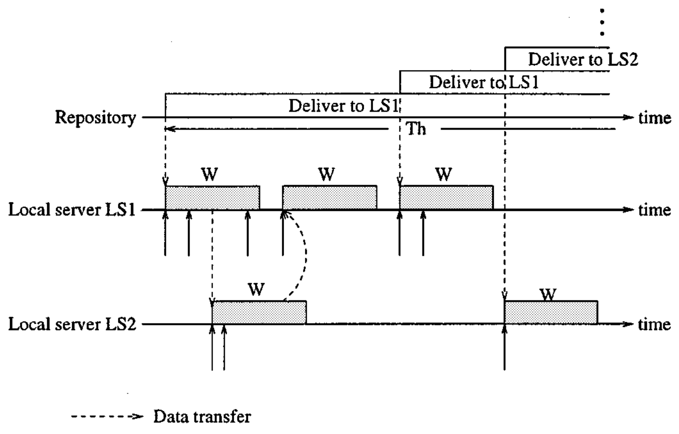


Fig. 6. Scheme for communicating servers.

be served locally. Another request for the same movie arriving at another local server (server LS_2) within the window obtains its data directly from server LS_1 instead of from the repository. LS_2 in turn buffers the data for a time W minutes. A later arrival in LS_1 can then be served from LS_2 by a "reverse" transfer. In this way, video data can be relayed from one server to another to form a "server chain" (the so-called "chaining"). The chain is "broken" and a new network stream is allocated when two successive buffer allocations in the system are offset in time by more than W minutes. Note that the buffers in the local servers are not shared (i.e., there is no united cache for all the servers).

IV. Scheme Analysis

In this section, we analyze the storage and channel requirements for each caching scheme. As mentioned before, it suffices to consider the single movie case. In the following, we focus on a movie of length T_h minutes, with streaming rate b_0 MB/min and its request process being Poisson. We are interested in the following performance measures.

- The average number of network channels (or concurrent streams) $\overline{S}$ used to serve the movie from the repository to the local servers.
- The average buffer requirement (in video minutes) for the movie in the local server i, $\overline{B}_i$ ($1 \leq i \leq N_s$, where N_s is the number of local servers in the system).
- The total system cost consisting of the channel cost and local storage cost (the streaming cost from the local servers to their peers and users is considered relatively insignificant and hence is ignored). For illustrative purpose, we consider linear storage and channel cost, though other functions can be as well used in our analysis. Let β be the cost in using a network channel in \$/(min $\cdot$ channel), and α be the storage cost in \$/(min·MB). The total system cost is the sum of the storage cost in all the local servers and the network channel cost. Given that there are N_s local servers, the total average cost is then $\alpha b_0 \sum_{i=1}^{N_s} \overline{B}_i + \beta \overline{S}$. Defining $\hat{C}$ as the normalized cost with respect to β and recalling that $\gamma \triangleq b_0\alpha/\beta$, we have

$$\hat{C} = \gamma \sum_{i=1}^{N_s} \overline{B}_i + \overline{S}. \tag{1}$$

TABLE II
NOMENCLATURE USED IN THIS PAPER

Symbol		Meaning
T_h	:	Movie length (minutes)
b_0	:	Streaming rate of the movies (MB/min)
α	:	Storage cost (\$/(min·MB))
β	:	Network channel cost (\$/(min·channel))
γ	:	$\triangleq b_0\alpha/\beta$, storage cost w.r.t. channel cost (channel/min, or simply, /min)
N_s	:	The number of local servers in the system
λ_i	:	Request rate for a specific movie in the local server i (req/min)
λ	:	Total request rate for a specific movie in the system (req/min)
Λ	:	Total request rate for all the movies in the system (req/min)
$\overline{B}_i$	:	Average buffer used for a movie in the local server i (minutes)
$\overline{S}$	:	Average number of repository channels used for a movie
C	:	The total cost of the distributed architecture (\$/min)
$\hat{C}$	:	Normalized total cost of the distributed architecture w.r.t. β

Obviously, for the true-VOD case, the buffer requirement is zero and $\overline{S}$ is given by the total arrival rate for the movie times T_h.

We summarize in Table II the symbols we use in this paper.

A. Unicast Delivery

In this case, we can focus on an arbitrary local server (and hence drop its index i). Requests for the movie arrive at the server with rate λ req/min. Clearly, the average interval between successive network channel allocations is given by $W + 1/\lambda$; therefore, the average number of channels used is, by Little's formula

$$\overline{S} = \frac{T_h}{W + 1/\lambda}. \tag{2}$$

Note that the buffer of size W minutes is held in the server for the duration of T_h minutes. Since such a buffer is allocated on average once every $W + 1/\lambda$ minutes, by Little's formula, the average number of buffers allocated is $(1/(W+1/\lambda))T_h$; hence, the average buffer size is

$$\begin{aligned}\overline{B} &= \frac{T_h}{W + 1/\lambda}W \\ &= \frac{\lambda W T_h}{1 + \lambda W}.\end{aligned} \tag{3}$$

We clearly see from (2) and (3) that $\overline{S} = \lambda T_h - \lambda\overline{B}$, which yields

$$\begin{aligned}\hat{C} &= \gamma\overline{B} + \overline{S} \\ &= (\gamma - \lambda)\overline{B} + \lambda T_h.\end{aligned} \tag{4}$$

Therefore, for linear cost functions, minimizing $\hat{C}$ means that we either do not cache at all (the case corresponding to $\lambda < \gamma$, making $\overline{B} = W = 0$) or we store the whole movie in the local server (the case corresponding to $\lambda \geq \gamma$, making $W = \infty$ and hence $\overline{B} = T_h$).

B. Multicast Delivery

Let the request rate for the movie in server i be λ_i req/min ($i = 1, \ldots, N_s$) and the total request rate of the movie in the system be given by $\lambda = \sum_{i=1}^{N_s} \lambda_i$.

- Prestoring: Server i has to store W minutes of movie permanently and, in addition, a variable buffer size depending on whether any request arrives within a multicast interval. If there is an arrival, a buffer of size W minutes is held for a time $T_h - W$ minutes. Since the probability of an arrival is $p = 1 - e^{-\lambda_i W}$, the rate of such a buffer allocation is p/W. Using Little's formula, the sum of the fixed and variable parts of the buffer amount is therefore

$$\begin{aligned}\overline{B}_i &= W + \frac{1 - e^{-\lambda_i W}}{W}(T_h - W)W \\ &= W + (1 - e^{-\lambda_i W})(T_h - W).\end{aligned} \tag{5}$$

Regarding $\overline{S}$, recall that a stream is held for a time $T_h - W$ minutes if there is an arrival within a multicast interval (with probability p). Using Little's formula again, we have

$$\overline{S} = \frac{1 - e^{-\lambda W}}{W}(T_h - W). \tag{6}$$

- Precaching: We first consider *periodic multicasting with precaching*. If there is no arrival within W minutes in a local server (with probability $e^{-\lambda_i W}$), an average buffer size of $W/2$ minutes would be held for W minutes; otherwise, a buffer size of W minutes is held for T_h minutes. Hence, the average storage used is

$$\begin{aligned}\overline{B}_i &= \frac{e^{-\lambda_i W}}{W}W\frac{W}{2} + \frac{1 - e^{-\lambda_i W}}{W}T_h W. \\ &= e^{-\lambda_i W}\frac{W}{2} + (1 - e^{-\lambda_i W})T_h.\end{aligned} \tag{7}$$

Regarding $\overline{S}$, if there is no request arriving within the multicast interval, the multicast stream would be aborted after W minutes; otherwise it would be used for a time T_h minutes. Therefore

$$\begin{aligned}\overline{S} &= \frac{1}{W}\left(e^{-\lambda W}W + (1 - e^{-\lambda W})T_h\right) \\ &= e^{-\lambda W} + (1 - e^{-\lambda W})\frac{T_h}{W}.\end{aligned} \tag{8}$$

Note that $\lim_{W\to 0}\overline{S} = \lambda T_h + 1$, which is one stream more than the true-VOD case. This is expected because when $W \to 0$, streams are "started and aborted" continuously and such wastage amounts to one full stream.

The decision problem on whether a server should precache or not can be formulated as follows. We associate with server i a decision variable x_i with $x_i = 1$ meaning that the server precaches and $x_i = 0$ otherwise. We minimize the system cost by solving the following for the multicast schedule W^* and variable x_i:

minimize

$$\gamma \sum_{i=1}^{N_s} \left(\frac{W}{2} e^{-\lambda_i W} + (1 - e^{-\lambda_i W}) T_h \right) x_i + \exp\left(-\sum_{i=1}^{N_s} \lambda_i x_i W \right) \left(1 - \prod_{i=1}^{N_s} (1 - x_i) \right) + \left(1 - \exp\left(-\sum_{i=1}^{N_s} \lambda_i x_i W \right) \right) \frac{T_h}{W} + \sum_{i=1}^{N_s} \lambda_i T_h (1 - x_i),$$

w.r.t.

$$W, x_1, x_2, \ldots, x_{N_s}$$

subject to

$$W \geq 0,\ x_i \in \{0, 1\},\ \text{for } i = 1, \ldots, N_s$$

where the first line of the objective function is the total storage cost, the second and third lines correspond to the total network channel cost due to precaching servers (the product term simply says that if all the servers do not precache, the overhead term given by the exponential does not exist), and the fourth line is the network channel cost due to those nonprecaching servers (i.e., the true-VOD case). Note that for uniform load (i.e., $\lambda_i = \lambda/N_s$), the solution of the above is greatly simplified: either all $x_i = 0$ (all servers do not precache) or $x_i = 1$ (all servers precache).

We now consider *request-driven precaching*, and that all servers precache (the decision problem whether a server precaches or not can be posed similarly as above). In this scheme, the average time between successive channel allocation is given by $W + 1/\lambda$, and hence (by Little's formula)

$$\overline{S} = \frac{\lambda T_h}{1 + \lambda W}. \tag{9}$$

In a local server, the precached data (of average amount of $W/2$ minutes) is flushed at the end of W minutes if no request arrives within that time (which occurs with probability $e^{-\lambda_i W}$) *and* the stream is not started by the server (which occurs with probability $1 - \lambda_i/\lambda$). Since the two events are independent, the probability that the precached data is flushed is given by

$$\pi_0 = e^{-\lambda_i W} \left(1 - \frac{\lambda_i}{\lambda} \right). \tag{10}$$

Therefore, the average buffer size used in the local server i is given by

$$\overline{B}_i = \frac{1}{W + 1/\lambda} \left[\pi_0 \frac{W^2}{2} + (1 - \pi_0) T_h W \right]. \tag{11}$$

C. Communicating Servers

We consider that there are many local servers (i.e., N_s reasonably large) so that a subsequent request likely comes from a different server. Since the average number of concurrent requests is λT_h, the average buffer requirement in the system is given by

$$\overline{B} = \lambda T_h W. \tag{12}$$

We next obtain $\overline{S}$. A request extends the chain if its arrival time falls within W minutes from the previous one, i.e., with probability $p = 1 - e^{-\lambda W}$. Therefore, the average interarrival time of the requests given that the successive requests are within W minutes is

$$T_a = \frac{1}{p} \int_0^W \lambda x e^{-\lambda x}\, dx = \frac{1}{\lambda} - \frac{W}{e^{\lambda W} - 1}. \tag{13}$$

Note that each such arrival extends the time which the video data stays in the network by, on average, T_a minutes. The average time data is kept in the server network is hence given by

$$L = \sum_{i=0}^{\infty} p^i (1 - p) i T_a + W = T_a e^{\lambda W} + W. \tag{14}$$

Therefore, the average interval between successive channel allocation is

$$T_s = L + \frac{1}{\lambda}. \tag{15}$$

By Little's formula, the average number of concurrent streams is given by

$$\overline{S} = \frac{T_h}{T_s}. \tag{16}$$

V. Illustrative Examples and Comparisons

In this section, we compare the optimal window size W^* given λ in order to minimize the system cost.

A. Unicast Delivery

Recall that in unicast delivery, the tradeoff between $\overline{S}$ and $\overline{B}$ given λ is linear with slope $-\lambda$, and for linear cost function the optimal caching strategy is either not storing the movie at all (corresponding to $W^* = 0$) or storing the movie completely (corresponding to $W^* = \infty$), depending on whether $\lambda < \gamma$ or not, respectively. We show in Fig. 7 the break-even request rate with respect to γ. If λ of a movie is higher than the break-even rate, the movie is stored completely in the local server; otherwise, true-VOD is used. We clearly see that when γ is low (i.e., storage cost is cheap compared to the channel cost) or λ is high

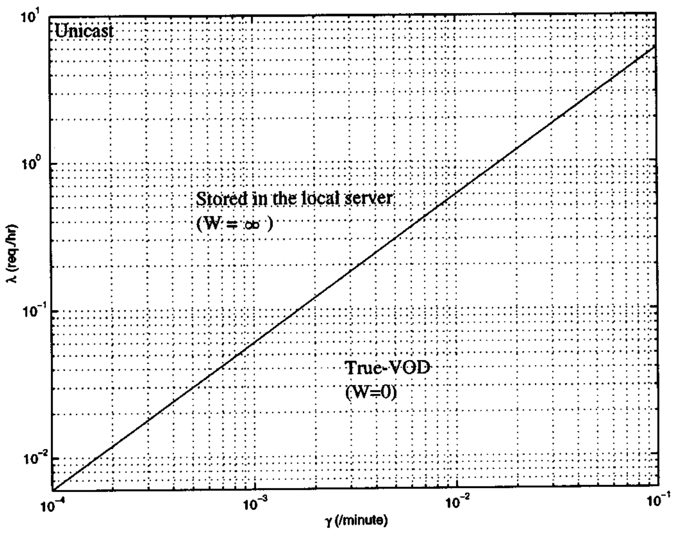


Fig. 7. Relationship between λ and γ to minimize the system cost for unicast delivery.

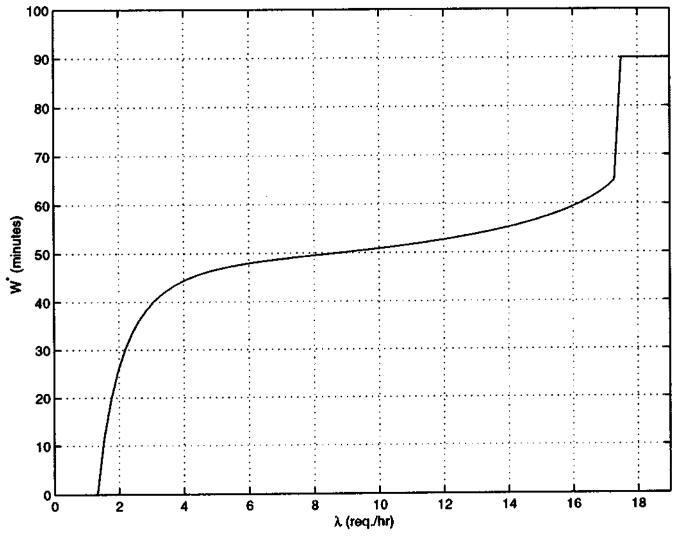


Fig. 9. W^* versus λ for prestoring ($N_s = 20, \gamma = 0.002$/min, and $T_h = 90$ min).

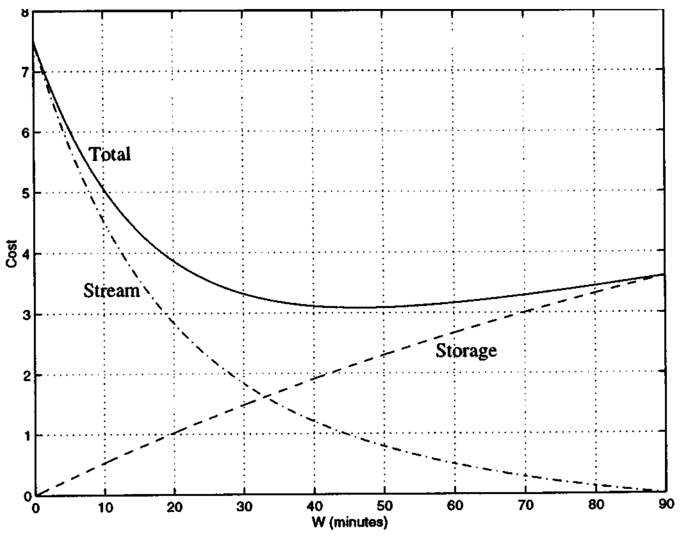


Fig. 8. $\hat{C}$, $\overline{S}$, and $\gamma\overline{B}$ versus W for prestoring ($\lambda = 5$ req/min, $N_s = 20$, $\gamma = 0.002$/min, and $T_h = 90$ min).

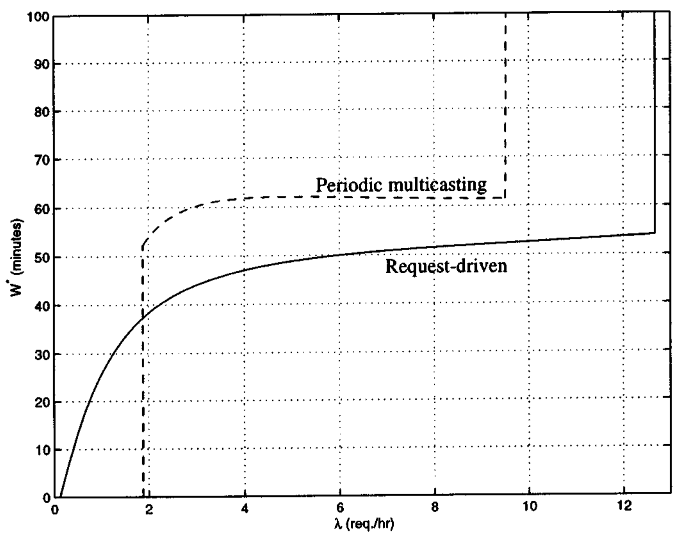


Fig. 10. W^* versus λ for precaching ($\gamma = 0.002$/min, $N_s = 20$, and $T_h = 90$ min).

(popular title), the movie is likely to be entirely stored at the local servers.

B. Multicast Delivery

We first consider prestoring. We show in Fig. 8 the normalized cost (with respect to β) of streaming, storage, and their total with respect to the buffer size W, with $N_s = 20$, $\lambda = 5$ req/min, $\gamma = 0.002$/min, $T_h = 90$ min, and uniformly loaded local servers. As W increases, the number of network streams required (and hence network cost) decreases. When $W = T_h$, the movie is entirely stored in the local server and hence there is no need of network channels to stream movies from the repository. On the other hand, as W increases, the storage cost increases from zero to a total of $\gamma N_s T_h$. Regarding the total cost $\hat{C}$, it first decreases quite sharply to a minimum and then rises slowly (the rise is more marked if γ is higher). Clearly, there is a W^* which minimizes $\hat{C}$. As γ increases (i.e., the storage cost becomes relatively more expensive), W^* decreases and $\hat{C}$ increases.

We show in Fig. 9 W^* as a function of λ, using the same parameters as before. As λ increases, W^* increases quite sharply and then remains quite flat. In the flat region, the movie is partially cached. There is a minimum λ below which prestoring is not worthwhile because of the low popularity of the movie (corresponding to the true-VOD case). There is also a λ beyond which the movie should be entirely stored at the local servers (corresponding to $W^* = 90$ min).

We next consider precaching. We show in Fig. 10 the optimal W^* as a function of λ for the cases of periodic multicasting and request-driven precaching (uniform server load $\lambda_i = \lambda/N_s$, $\gamma = 0.002$/min, and $T_h = 90$ min). As before, there is a λ below which there should be no precaching at all (corresponding to $W^* = 0$, the true-VOD case) and a λ above which the movie should be entirely stored (corresponding to $W^* = \infty$). For λ

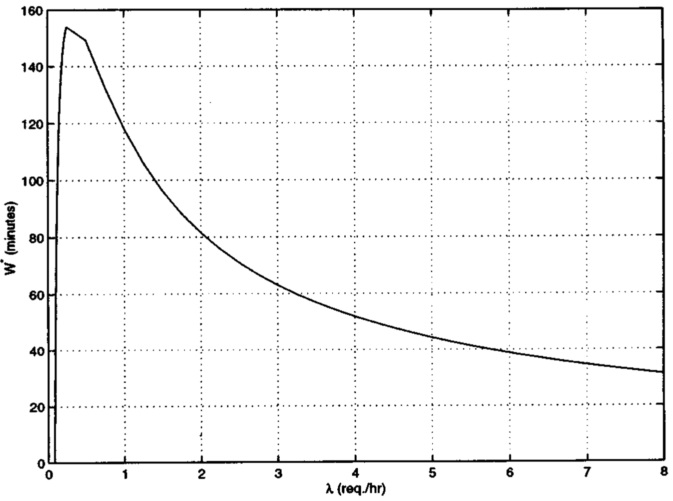


Fig. 11. W^* versus λ for chaining ($\gamma = 0.002$/min, and $T_h = 90$ min).

somewhere in between, the movie is partially cached with W^* remaining quite constant.

C. Communicating Servers

We plot in Fig. 11 W^* versus λ for communicating servers, with $T_h = 90$ min and $\gamma = 0.002$/min. Again, we see that when λ is low, the movie should be directly streamed from the repository (i.e., the true-VOD case). As λ increases, W^* first increases rather sharply so that requests can be chained more effectively, and then decreases due to the fact that the higher request rate makes a chain easier to be formed, leading to a lower buffer requirement.

We finally compare the optimal cost for all the schemes we have studied by showing in Fig. 12 their $\hat{C}^*$ (corresponding to $W = W^*$) versus λ, with $\gamma = 0.002$/min, and $T_h = 90$ min. Also shown is the case of true-VOD, whose cost increases linearly with λ. We have considered for simplicity that the costs of unicast and multicast channels are the same. (Note that they certainly do not have to be so and this can be easily taken into account given our analysis.) All our schemes achieve lower cost than true-VOD. For unicast delivery, the cost first follows that of true-VOD ($W^* = 0$) and then flattens off at the point at which the movie is stored completely at the local servers. We see that multicasting can achieve substantial cost reduction when compared with the unicast case only if the request rate is within a certain (narrow) window of λ. If multicast streams are more expensive than unicast streams, the cost advantage of multicasting would be further reduced. Compared with all the other schemes, chaining achieves significantly lower cost. Even if there were a cost associated with interserver communication, the cost of chaining is not likely to be higher than the other systems (unless such communication cost is very high).

VI. Cost Comparison with a System using Request Batching and Multicasting

In order to increase the user capacity of the repository, request batching and multicasting has been proposed, in which requests for a movie arriving within a period of time are grouped or "batched" together and served with a single multicast stream.

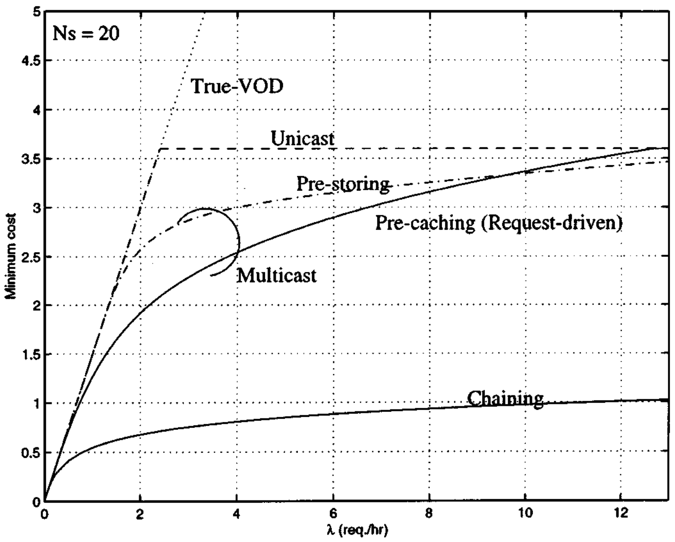


Fig. 12. Comparison of $\hat{C}^*$ versus λ for the proposed caching schemes ($\gamma = 0.002$/min, and $T_h = 90$ min).

(See [2], [25], [26] and references therein for more details.) If users can tolerate a certain maximum delay of $D_{\max}$ minutes, the batching period can be set to be no more than $D_{\max}$ minutes. We may use a batching scheme in which the first user following a movie showing starts a batching window of size $W = D_{\max}$, at the end of which the movie is multicast to all the requests arriving within the window. By Little's formula, the average number of streams required given the movie's request rate λ, and hence the normalized total system cost (because there is no storage cost), is $\overline{S} = \hat{C} = T_h/(1/\lambda + D_{\max})$ [26].

A distributed servers architecture achieves negligible user delay by means of streaming servers. This advantage, however, comes with the cost of additional servers and storage. However, when compared with request batching in which the repository directly multicast movies to the requests, a distributed servers architecture may achieve even lower total system cost depending on the acceptable maximum user delay and the storage cost. This is illustrated as follows.

Let us first consider a distributed servers architecture using the unicast caching scheme. Let N_s be the number of servers in the system, with the request rate at server i for a movie being λ_i, with $\sum_{i=1}^{N_s} \lambda_i = \lambda$. The minimum cost as discussed before can be expressed as

$$\hat{C}^* = \sum_{i=1}^{N_s} [\gamma T_h u(\lambda_i - \gamma) + (1 - u(\lambda_i - \gamma))\lambda_i T_h] \qquad (17)$$

where $u(x)$ is the unit-step function with $u(x) = 1$ for $x \geq 0$ and $u(x) = 0$ otherwise.

We plot in Fig. 13 the cost of the distributed servers architecture versus $\lambda = N_s\lambda_i$ given a certain value of $N_s\gamma$. Also shown is the cost of a system in which the repository multicasts movies directly to the requests (i.e., the "request batching" system), with $D_{\max} = 6$ min (we have considered that the cost of unicast channels is the same as that of the multicast channels). For values of $N_s\gamma$ higher than a certain value ($= 1/D_{\max}$), the distributed servers architecture indeed has a higher cost, hence trading off system cost with lower (zero) user delay. However,

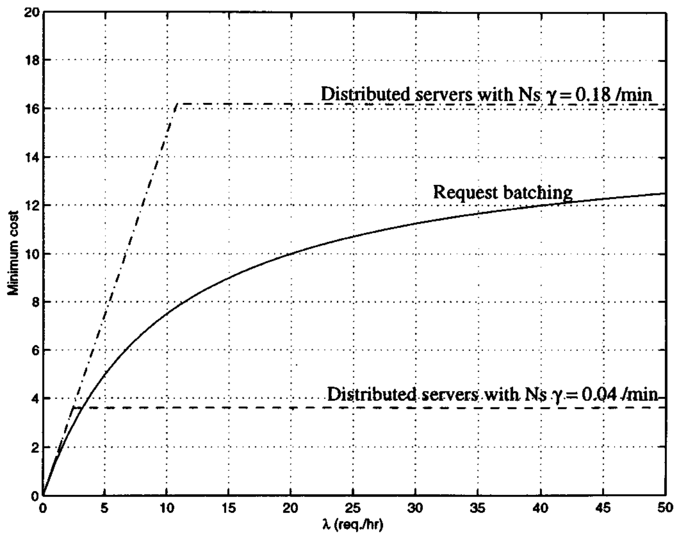


Fig. 13. Comparison of $\hat{C}^*$ versus λ between a system with request batching and a distributed servers architecture based on unicast delivery ($D_{\max} = 6$ min, and $T_h = 90$ min).

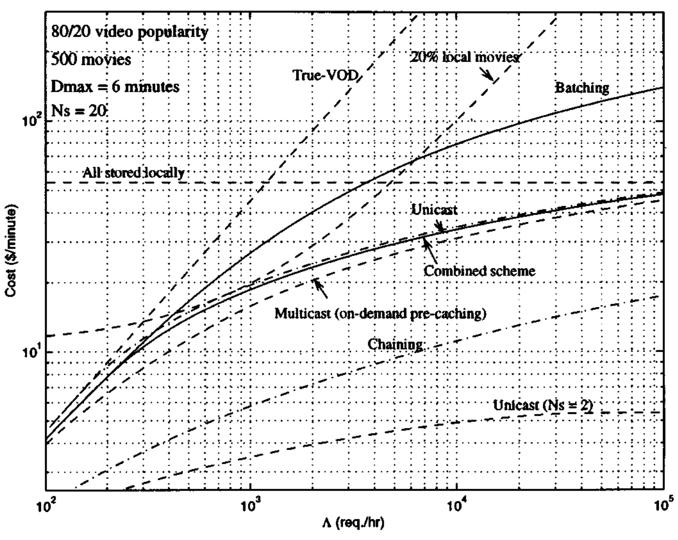


Fig. 14. Total cost per minute versus Λ for different systems ($\gamma = 0.002$/min, $\beta = \$0.03$/(min·channel), $N_s = 20$, number of movies = 500, 80/20 video popularity, $D_{\max} = 6$ min, and $T_h = 90$ min).

with a low enough $N_s\gamma$ ($< 1/D_{\max}$), the cost lines of both systems cross at

$$\lambda' = \frac{N_s\gamma}{1 - N_s\gamma D_{\max}}. \qquad (18)$$

From the above, we see that if many movies have λ higher than λ', the distributed servers architecture as compared to a batching system can achieve both lower cost *and* lower delay to the users. This is because distributed servers architecture can effectively trade off the (cheaper) storage cost with the (more expensive) network channel cost. To achieve an even lower system cost, we can use a "combined" scheme: when $\lambda < \lambda'$ we use request batching at the repository to deliver movies to users, and for $\lambda > \lambda'$ we store the movie at the local server.

As an illustrative example on the cost advantage of a distributed servers architecture, let us consider a system with 500 movie titles of heterogeneous video popularity of 80/20 geometric movie popularity (refer to Fig. 1) with aggregate arrival rate Λ req/h, with $\beta = 0.03$/min and $\gamma = 0.002$/min (corresponding to the second column in Table I). We consider four "traditional" systems for comparison:

1) *True-VOD*: There are no local servers and the repository serves all the requests on the demand basis.
2) *Batching*: There are no local servers and the repository uses request batching and multicasting to serve all the requests with maximum user delay $D_{\max}$.
3) *All stored locally*: A system with N_s local servers, each of which replicates all the movies and serves Λ/N_s req/min.
4) *20% local movies*: A system with a repository and N_s uniformly loaded local servers, each of which stores the most popular 20% of the movies.

We compare the cost of the above systems with the distributed servers architecture proposed in this paper, namely:

1) *Unicast*: A system with a repository and N_s uniformly loaded local servers using unicast delivery.
2) *Combined scheme*: A system of a repository and N_s uniformly loaded local servers, with the local servers storing movies with $\lambda \geq \lambda'$, and the repository using batching and multicasting to deliver the rest of the movies.
3) *Multicast* (request-driven precaching): A system using request-driven precaching with N_s uniformly loaded servers.
4) *Chaining*: A chaining system with many communicating servers.

We show in Fig. 14 the cost C versus the aggregate request rate Λ, for the systems ($N_s = 20$, $D_{\max} = 6$ min, and $T_h = 90$ min). From the cases of true-VOD and storing all the movies locally, we see that when Λ is low, we should use true-VOD, and when Λ is high, we should replicate all the movies locally. Batching reduces the cost substantially as compared to the true-VOD case. However, when Λ is high, batching actually has a higher cost when compared with replicating all the movies locally (because batching cannot take advantage of the low storage cost). Storing the more popular set of movies locally achieves good cost saving for a certain range of Λ, showing that naively storing 20% of the most popular set of movies in local servers does not always achieve good cost saving.

Our unicast case and combined scheme achieve similarly low cost. This is expected because batching is not effective for those not-so-popular movies. Using multicast achieves further reduction in cost. However, the reduction in cost is not so significant, mainly because, as shown before, multicasting achieves lower cost only for a certain (narrow) range of movie request rate (while in this example, the request rate of the movies differs by orders of magnitude). Chaining achieves by far the greatest saving in cost. We see that distributed servers architecture can achieve lower cost by more than an order of magnitude as compared with the true-VOD case. It achieves significantly lower cost than a batching system *while* offering on-demand services. Such cost advantage is greater when the video popularity is more skewed. As N_s decreases, the cost of distributed servers architecture decreases further because less data is replicated.

VII. Conclusion

In a video system, a number of repository servers store all the video contents of interest to a large number of geographically distributed users. Due to the limited repository capacity and likely high transmission cost from the repository to users, using only the repository to serve the population would not be cost effective. A better solution is to use a distributed servers architecture, in which local servers are put close to user clusters and cache video contents according to their local demands. By increasing the number of repository and local servers, this architecture achieves storage and streaming scalabilities.

We have studied a number of local caching schemes to offer networked on-demand video services. The caching schemes pertain to whether the repository is able to multicast data to the local servers or not, and whether the local servers can exchange video contents with each other or not. All of our caching schemes keep a circular buffer for each movie being displayed and hence the movie is partially cached. The schemes are able to reduce the network bandwidth used and, as compared with treating a movie as a single entity, achieves much better tradeoff between network channels with local storage (and hence lower overall system cost). We have studied the tradeoff between the number of network channels required and the local storage required for each scheme. Given certain cost functions, we have addressed how much to cache in order to minimize the total system cost consisting of the costs of local storage and repository streams.

Generally, the unpopular movies should be streamed directly from the repository, and the popular ones should be replicated entirely at the local servers. Movies of intermediate popularity are likely to be partially cached in the local servers. Multicasting is able to offer substantial reduction in system cost only for a certain range of movie request rate. If the local servers can exchange data with each other, the system cost can be significantly reduced. We show that, given today's low storage cost, distributed servers architecture effectively trades off storage cost with communication cost. As compared with a system using request batching and multicasting, distributed servers architecture can achieve both lower overall system cost and lower user delay. The more skewed the video popularity is, the more saving a distributed servers architecture can achieve.

References

[1] T. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia Mag.*, pp. 14–24, Fall 1994.

[2] V. O. K. Li and W. Liao, "Distributed multimedia systems," *Proc. IEEE*, vol. 85, pp. 1063–1108, July 1997.

[3] F. A. Tobagi, "Distance learning with digital video," *IEEE Multimedia Mag.*, pp. 90–94, Spring 1995.

[4] A. D. Gelman, H. Kobrinski, L. S. Smoot, S. B. Weinstein, M. Fortier, and D. Lemay, "A storeand-forward architecture for video-on-demand service," *Can. J. Electr. Comput. Eng.*, vol. 18, pp. 37–40, Jan. 1993.

[5] 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, Aug. 1996.

[6] L. D. Giovanni, A. M. Langellotti, L. M. Patitucci, and L. Petrini, "Dimensioning of hierarchical storage for video on demand services," in *Proc. IEEE ICC*, 1994, pp. 1739–1743.

[7] Y. N. Doğanata and A. N. Tantawi, "Making a cost-effective video server," *IEEE Multimedia Mag.*, pp. 22–30, Winter 1994.

[8] F. Schaffa and J.-P. Nussbaumer, "On bandwidth and storage tradeoffs in multimedia distribution networks," in *Proc. Infocom*, 1995, pp. 1020–1026.

[9] P. Lie, J. Lui, and L. Golubchik, "Threshold-based dynamic replication in large-scale video-on-demand systems," in *Proc. 8th Int. Workshop Research Issues in Data Engineering*, Orlando, FL, Feb. 1998, pp. 52–59.

[10] T. D. C. Little and D. Venkatesh, "Popularity-based assignment of movies to storage devices in a video-on-demand system," *ACM/Springer Multimedia Syst.*, no. 2, pp. 280–287, 1995.

[11] C. Papadimitriou, S. Ramanathan, P. V. Rangan, and S. SampathKumar, "Multimedia information caching for personalized video-on-demand," *Comput. Commun.*, vol. 18, pp. 204–216, Mar. 1995.

[12] Y. Wang, Z.-L. Zhang, D. H. C. Du, and D. Su, "A network-conscious approach to end-to-end video delivery over wide area networks using proxy servers," in *Proc. IEEE Infocom*, San Francisco, CA, Mar. 29–Apr. 2, 1998, pp. 660–667.

[13] A. Dan, M. Kienzle, and D. Sitaram, "A dynamic policy of segment replication for load-balancing in video-on-demand servers," *Multimedia Syst.*, vol. 3, pp. 93–103, July 1995.

[14] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Syst.*, vol. 4, pp. 197–208, Aug. 1996.

[15] L.-S. Juhn and L.-M. Tseng, "Enhanced harmonic data broadcasting and receiving scheme for popular video service," *IEEE Trans. Consumer Electron.*, vol. 44, pp. 343–346, May 1998.

[16] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "Design and analysis of permutation-based pyramid broadcasting," *ACM/Springer Multimedia Syst.*, vol. 7, no. 6, pp. 439–448, 1999.

[17] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," *ACM Comput. Commun. Rev.*, vol. 27, pp. 89–100, Oct. 1997.

[18] L. Gao, J. Kurose, and D. Towsley, "Efficient schemes for broadcasting popular videos," in *Proc. NOSSDAV*, Cambridge, U.K., July 1998.

[19] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, New York, NY, Sept. 14–16, 1998, pp. 191–200.

[20] J.-F. Pâris, S. W. Carter, and D. D. E. Long, "A hybrid broadcasting protocol for video on demand," in *Proc. 1999 IS&T/SPIE Conf. Multimedia Computing and Networking*, San Jose, CA, Jan. 1999, pp. 317–326.

[21] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, Ottawa, Canada, June 1997, pp. 110–117.

[22] J. Y. B. Lee, "UVOD—A unified architecture for video-on-demand services," *IEEE Commun. Lett.*, vol. 3, pp. 277–279, Sept. 1999.

[23] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia Mag.*, pp. 51–62, Oct.–Dec. 1997.

[24] S.-H. G. Chan and F. A. Tobagi, "Caching schemes for distributed video services," in *Proc. IEEE Int. Conf. Communications (ICC'99)*, Vancouver, BC, Canada, June 1999, pp. 994–1000.

[25] ——, "On achieving profit in providing near video-on-demand services," in *Proc. IEEE Int. Conf. Communications (ICC'99)*, Vancouver, Canada, June 1999, pp. 988–994.

[26] S.-H. G. Chan, F. A. Tobagi, and T.-M. Ko, "Providing on-demand video services using request batching," in *Proc. IEEE Int. Conf. Communications (ICC'98)*, Atlanta, GA, June 1998, pp. 1716–1722.

S.-H. Gary Chan (M'98) received the Ph.D. degree in electrical engineering with a minor in business administration from Stanford University, Stanford, CA, in 1999, and the B.S.E. degree (Highest Honor) in electrical engineering from Princeton University, Princeton, NJ, in 1993.

He is currently an Assistant Professor with the Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, and an Adjunct Researcher with Microsoft Research China, Beijing. He was a Visiting Assistant Professor in networking at the University of California, Davis, from September 1998 to June 1999. During 1992–1993, he was a Research Intern at the NEC Research Institute, Princeton. His research interest includes multimedia networks, services, and systems, high-speed and wireless communications networks, and Internet technologies and protocols.

Dr. Chan was a William and Leila Fellow at Stanford University during 1993–1994. At Princeton, he was the recipient of the Charles Ira Young Memorial Tablet and Medal, and the POEM Newport Award of Excellence in 1993. He is a member of Tau Beta Pi, Sigma Xi, and Phi Beta Kappa.

Fouad Tobagi (M'77–SM'83–F'85) received the Engineering degree from Ecole Centrale des Arts et Manufactures, Paris, France, in 1970 and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles, in 1971 and 1974, respectively.

From 1974 to 1978, he was a Research Staff Project Manager with the ARPA project at the Computer Science Department, University of California, Los Angeles, and engaged in research in packet radio networks, including protocol design, and analysis and measurements of packet radio networks. In 1978, he joined the faculty of the School of Engineering at Stanford University, Stanford, CA, where he is Professor of electrical engineering and computer science. In 1991, he co-founded Starlight Networks, Inc., a venture concerned with multimedia networking, and served as Chief Technical Officer until 1998. His research interests include packet switching in ground radio and satellite networks, high-speed local area networks, fast packet switching, broadband integrated services digital networks, asynchronous tranfer mode, multimedia networking and communications, and modeling and performance evaluation of network systems. He has served as Associate Editor for Computer Communications for the IEEE TRANSACTIONS ON COMMUNICATIONS from 1984 to 1986, Editor for Packet Radio and Satellite Networks in the *Journal of Telecommunications Networks* from 1981 to 1985, Co-Editor of the special issue on local area networks of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (November 1983), Co-Editor of the special issue on packet radio networks of the PROCEEDINGS OF THE IEEE (January 1987), and Co-Editor of the special issue on large scale atm switching systems for B-ISDN of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (October 1991). He has also served as co-editor of *Advances in Local Area Networks*, a book in the series *Frontiers in Communications* (New York: IEEE Press). He is currently serving as editor for a number of journals in high-speed networks, wireless networks, multimedia, and optical communications.

Dr. Tobagi is a member of the Association for Computing Machinery and served as an ACM National Lecturer from 1982 to 1983. He was the winner of the 1981 Leonard G. Abraham Prize Paper Award in the field of communications systems for his paper "Multiaccess Protocols in Packet Communications Networks," and co-winner of the IEEE Communications Society 1984 Magazine Prize Paper Award for the paper "Packet Radio and Satellite Networks." He was co-recipient of the 1998 Kuwait Prize in the field of applied sciences.