Providing Distributed On-Demand Video Services Using Multicasting and Local Caching

S.-H. Gary Chan Fouad A. Tobagi
Department of Electrical Engineering
Stanford University
Stanford, CA 94305

Abstract— **In an on-demand video system, a number of repository servers store all the video contents of interest to a large pool of geographically distributed users. In order to achieve higher streaming capacity and lower long-haul transmission cost, a distributed servers architecture can be used, in which multiple local servers are placed close to the users and cache the video contents dynamically according to their local demands. We consider in this paper that the repository delivers video contents to the local servers using multicasting. Upon the arrival of a request the local server joins the on-going multicast streams of the movie and caches the data (i.e., "latches" onto the streams), while a transient network stream is allocated to supply the missing (beginning) portion of the movie due to the time-lag between the request and the multicast streams. After a short while, all the streams will "merge" into one stream. Such scheme is hence called "latching and merging." We study the required number of streams and local buffer for this scheme, the effect of the maximum number of streams that a local server can latch onto, and its advantages in providing distributed on-demand video services.**

Keywords— **On-demand video services, distributed servers architecture, multicasting, local caching, network channels**

I. Introduction

Advances in computer and networking technologies have made on-demand video services (video-on-demand (VOD)) a reality. Important video applications pertaining to entertainment, education, information, and advertising include movie-on-demand, distance learning, interactive news, home shopping, etc. [1], [2], [3], [4], [5], [6], [7], [8]

In an on-demand video system, a number of repository servers (collectively referred to as a repository) store all the video contents of interest to a large number of geographically distributed users. To achieve large and scalable storage, the repository servers may be tertiary libraries or jukeboxes. If videos were to be streamed directly to the users, the user capacity would be limited by the streaming capacity of the repository servers. The capacity can be increased by using a hierarchy of servers, in which multiple streaming servers cache locally the movies delivered from the repository while streaming them to the users. If the streaming servers are co-located with the repository, the transmission cost incurred in streaming videos to the remote users may be high. To overcome this problem and to take advantage of the locality of access, the streaming servers may be placed close to the user clusters, thus forming a distributed servers architecture. Such a system in fact has been discussed previously [3], [9], [10]. By increasing the number of repository servers and local servers, the system offers scalable storage and streaming capacities in providing distributed video services.

A service provider may set up a number of local servers each serving a region. If multicast network channels are available (through, for example, the use of an internet or a satellite system), they can be used to deliver videos to the local servers (so as to further decrease the network channel requirement and channel cost). If the local servers are connected by a network, they can also exchange their data with their peers. Under some circumstances, however, such exchange may not be feasible due to, for example, limited server processing capability (for constant and frequent content exchanges and updates), or network limitation (e.g., limited network capacity or disjoint networks). We show in Fig. 1 an example of such a case for a cable TV system. Multiple head-end servers serve their local communities through coaxial local drops. The repository may deliver the videos to the local servers using multicast channels. A local server hence can get its data from the repository by joining any existing multicast groups (so as to save a network stream from the repository). The multicast channels may be leased to the provider (and hence are available up to a fixed number), or can be acquired on demand. In this paper, we study the use of multicasting and local caching for video content delivery from the repository to the local servers (the cases of unicast channels and communicating servers have been studied in [10]).

The caching scheme we study allocates at the local server a certain amount of buffer corresponding to W minutes of a movie being displayed. The buffer is a shifting window, and hence future local requests arriving within the window may be served directly from the local cache, hence saving network channels/streams. Note that the caching scheme is able to keep a portion of a movie locally (i.e., *partial* movie caching), thereby achieving the trade-off between channels and storage. Keeping a portion of a movie locally also offers certain degree of user interactivity with the movies. We consider that the multicast network channels can be acquired on-demand and the number of such channels is unlimited. Accordingly, the servicing of requests pertaining to a given movie is independent of the servicing of requests for other movies, and hence it is sufficient to focus on a particular movie (i.e., the single movie case). We also

This work was supported in part by a grant from the Center for Telecommunications at Stanford University.

S.-H. Gary Chan was with Stanford University when this work was done. He is currently with the Department of Computer Science, University of California, Davis, CA 95616.

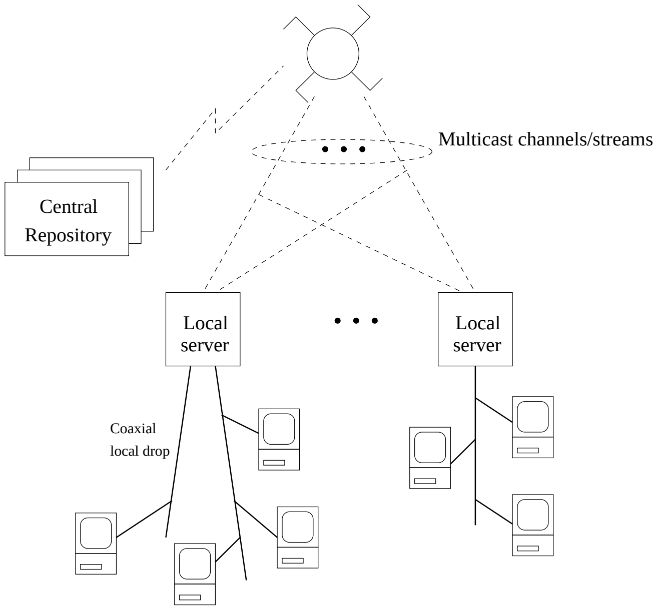

Fig. 1. A cable system using multicasting to deliver videos.

consider that the latency in the central repository is low and hence can be ignored.

This paper is organized as follows. After a brief review of previous work on distributed servers architecture in Sect. II, we describe in Sect. III the multicasting and local caching scheme we study in this paper (namely "latching and merging"). In Sect. IV, we analyze the scheme in terms of storage and channel requirements, while in Sect. V we provide some numerical results. We conclude in Sect. VI.

II. Previous Work

Barnett and Anido in [9] compare the 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. Giovanni et al. consider a hierarchical storage system in which local servers get their videos from a central repository. They study how the total cost of the storage and bandwidth can be minimized given certain goal in the user blocking probability [11]. The channels are assumed to be fixed and leased (while our work discusses on-demand channels). Schaffa and Nussbaumer study a VOD system in which servers are configured as a hierarchical balanced tree, and all video files are replicated at a certain level in the tree [12]. Trade-off between bandwidth cost and storage cost in a server is addressed. While the paper studies the case that the request rates to the various servers at a given level of the tree are the same, we consider here that the request rate to the servers may be different from each other. Little and Venkatesh consider in [13] a video system in which video files are replicated in order to decrease user loss rate. Lie et al. study a distributed VOD system in which movies are dynamically replicated and replaced in order to reduce user loss rate [14]. All the work above treat a video file as a single unit, and hence a movie is either completely stored (or replicated) in a local server or not at all. Here we consider schemes in which a movie can be *partially* cached, and hence they are able to achieve better trade-off between storage and network bandwidth.

Papadimitriou et al. in [15] consider a VOD system in which a central repository delivers videos through a series of caching nodes, given a prespecified viewing schedule. The scheduler optimally determines when, where and how long a file should be cached in a storage node, so that the total cost can be minimized. Our work differs from this in that we consider on-demand services here. Wang et al. consider transmitting movies using constant bit rate. By storing some of the "bursty" data of a movie in a local server, the requirement in the network bandwidth can be reduced (hence achieving trade-off in local storage and network bandwidth) [16]. Our work does not assume *a priori* knowledge of video bandwidth profile as in this work.

Ghafir and Chadwick in [17] study a hierarchical VOD system in which a central repository serves a number of local servers. Files are transferred from the repository to the servers before they are streamed. Popular movies may be replicated in the local servers to increase the system capacity. Dan et al. in [18] consider load-balancing issues among multiple disks in a server. A movie stored in a disk is divided into a number of segments; and depending on the load on a disk, the segment can be dynamically replicated to one of the other disks. When disk storage runs out, the replicated segments may be deleted. Though partial movie storage is considered, network bandwidth issues (its requirement and trade-off with the storage) have not been addressed.

We differ from all the above work in that we consider using multicasting and local caching to deliver video data to the local servers. Our schemes are quite similar to those mentioned in [19], [20]. However, the previous work focuses on offering user interactive capabilities for an on-going movie, while we are interested in providing on-demand video services (i.e., no initial start-up delay), and hence the context and the performance measures are different. We also consider a scheme in which a local server can latch on more than one network channel/stream, which has not been discussed before. While simulation results are presented in [19], [20], we provide analysis for the schemes presented in this paper.

III. Caching Scheme: Latching and Merging

In this scheme, the repository delivers a movie through multicast, and a local server allocates a sliding window of size W minutes for a movie being displayed. A request for the same movie within W minutes of the movie's multicast from a different server can join the on-going multicast stream and caches its data (i.e., "latches" onto the on-going multicast stream), while at the same time a transient stream is used to recover the "time lag" between the time at which the request arrives and the multicast stream. After a while (within W minutes), the transient stream will catch up with the multicast and be relinquished (i.e., "merged" with the original stream). Such a multicasting and local caching scheme is called "latching and merging." We consider such a scheme using periodic multicasting or on-demand multicasting in the following subsections. (Another variant of the multicasting and local caching scheme is that a local server pre-caches a multicasting movie in anticipation of its future need even though there may not be request for such a movie at that time. This scheme trades off additional local storage with transient streams and will be the subject of our future study.)

A. Periodic multicasting

In periodic multicasting, the repository multicasts a movie at regular intervals of $W = T$ minutes. Such multicast streams are called the *basic* streams, with which all requests will merge. Suppose initially there is no movie cached in a local server. Upon the arrival of a request, the server immediately latches on a current multicast stream of the movie, while at the same time a transient network stream is allocated to recover the missing (beginning) portion of the movie. Since both streams are cached, after a period of time ΔT equal to the time from the start of the current regular multicast to the arrival time of the request, the movie can be displayed directly from the cached data of the basic stream; by then the transient stream can be deallocated and the request is merged onto the basic stream (with the movie display point and the multicast point separated by ΔT). In this case, the basic stream will be held for the entire duration of the movie. If there is no request by the end of a multicast interval, the repository aborts the basic stream and a new one started.

B. On-demand multicasting

Periodic multicasting wastes some channels if there is no request arriving in the system during the multicast interval. On-demand multicasting operates the same way, except that multicast streams are started on a demand basis upon the arrival of a request. Again suppose an initially empty system. The arrival of a request at a local server starts a basic stream (a multicast stream held for the entire duration of the movie); the server then caches the movie with a sliding window of size W minutes in anticipation of its future local requests. If another request for the same movie arrives at a different server within W minutes, the server latches on the basic stream and requests for a transient stream which, after a period of time (within W minutes), is merged with the basic stream. Therefore, all transient streams started within W minutes of the basic stream will be eventually merged with the basic stream. A new basic stream has to be started for the first arrival more than W minutes from the previous basic stream.

Note that as the sliding window size W increases, requests are more likely to latch onto and merge with the basic stream, and hence the number of network channels decreases initially (while the storage requirement increases). However, as W is further increased, the length of the transient streams becomes longer due to the later requests (recall the ΔT above), and hence the total number of channels increases. Therefore, we should not operate the system beyond the W at which the network channels begin to increase.

Certainly transient streams do not have to be unicast streams — they themselves can also be multicast streams. If so, a local server can latches on multiple on-going multicasts and merge with the basic multicast stream sooner (leading to lower network channel requirement). Due to reasons such as simplicity or finite processing capacity of local servers, one may place a maximum number R of the multicast streams that a server may latch onto at one time in addition to its own transient stream ($R \geq 0$). Clearly, $R = 0$ corresponds to no latching and merging, while $R = \infty$ means that all the on-going multicast streams can be latched onto simultaneously. For finite R (≥ 1), we consider that a local server would latch onto a maximum of $R-1$ streams (all of which certainly are transients) following the basic stream.

IV. Analysis

We consider a movie of length T_h minutes, with the aggregate request process at the repository being Poisson with rate λ (req./min.).[1] Let the request rate at the local server i be λ_i; given that there are N_s servers in the system, clearly $\lambda = \sum_{i=1}^{N_s} \lambda_i$. We study the stream requirement for the above schemes in terms of the average number of channels (or concurrent streams) used for the movie, $\overline{S}$, and the average amount of storage for the movie in the local server i (in minutes of video time), $\overline{B}_i$. The total storage in the system is hence given by $\overline{B} = \sum_{i=1}^{N_s} \overline{B}_i$.

A. Periodic multicasting

The average number of streams used in this scheme is the average number of basic streams, $\overline{S}_b$, *plus* an additional term due to transient streams, $\overline{S}_{trans}$, i.e., $\overline{S} = \overline{S}_b + \overline{S}_{trans}$.

We first obtain $\overline{S}_b$. Note that if there is no request within the multicast interval W (with probability $e^{-\lambda W}$), the multicast stream would be used for only W minutes before it is aborted; otherwise it would be held for a time T_h minutes. Therefore,

$$\overline{S}_b = \frac{1}{W}\left(We^{-\lambda W} + (1 - e^{-\lambda W})T_h\right). \qquad (1)$$

To obtain $\overline{S}_{trans}$, let's consider the case $R = 1$ (the case for $R \geq 1$ will be studied for the on-demand multicasting). Given that n (≥ 1) requests arrive at the local server i ($i = 1, 2, \ldots, N_s$) within the multicast interval W, the earliest request would be on average $W/(n+1)$ minutes from the start of the current multicast, which is also the length of the transient stream for all the n users. Therefore, upon the removal of the condition on n, the average number of transient streams for server i is given by $(1/W)\sum_{n=1}^{\infty}(W/(n+1))(\lambda W)^n e^{-\lambda_i W}/n! = (1-(1+\lambda_i W)e^{-\lambda_i W})/(\lambda_i W)$. Summing over the N_s servers, the average number of streams is hence given by,

$$\overline{S} \;=\; e^{-\lambda W} + (1 - e^{-\lambda W})\frac{T_h}{W} +$$

[1] λ is expressed in req./hr in our plots.

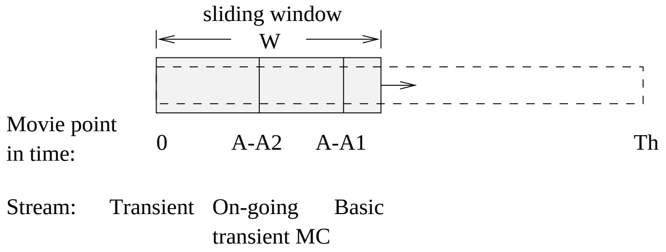

Fig. 2. Movie caching at a local server right after a request arrives at time A, showing that $M = 2$ streams start to fill the buffer besides the transient stream.

$$\sum_{i=1}^{N_s} \frac{1}{\lambda_i W} \left(1 - (1+\lambda_i W)e^{-\lambda_i W}\right). \qquad (2)$$

Regarding the buffer requirement at server i, *given* that there is an arrival within a multicast interval, a buffer size of W minutes would be allocated [2] for a time T_h minutes (yielding an average buffer size for the multicast interval equal to $WT_h/W = T_h$ minutes); otherwise, no buffer is used. Therefore, $B_i = (1 - e^{-\lambda_i W})T_h$, and

$$\overline{B} = \sum_{i=1}^{N_s}(1 - e^{-\lambda_i W})T_h. \qquad (3)$$

B. *On-demand multicasting*

We first give the ending time of a transient stream. Let A be the arrival time of a request. Let M be the number of on-going multicast streams that the local server simultaneously latches on ($M \leq R$) at that time, with the streams' starting times being ordered chronologically as $A_1, A_2, \ldots, A_M$ (A_1 is hence the start-time of the basic stream, while $A_2, \ldots, A_M$ are the start-time of the transient streams in the system). (As stated before, we need $A - A_1 \leq W$.) We show in Fig. 2 the scenario at the local server right after the request arrives, for $M = 2$. A sliding window of size W minutes is allocated, and the transient stream is supplying the current missing data, while the local server latches on two other multicast streams currently delivering movie data at movie time $A - A_2$ (transient multicast) and $A - A_1$ (basic stream). Once the buffer is filled, it will start to shift to the right.

From the figure, we clearly see that the transient stream for the local server will catch up with its closest stream, the Mth stream, after a time $(A - A_M)$; therefore, its ending time is initially scheduled to be

$$\begin{aligned} E_{M+1} &= A + (A - A_M) && (4)\\ &= 2A - A_M && (5) \end{aligned}$$

Since the existing M multicast streams (indexed by $m = 1, \ldots, M$) have to be long enough to fill the missing data in the "pipe" and it takes $(A_m - A_{m-1})$ minutes for the mth stream to catch up and merge with the $(m-1)$th stream ($m = 2, \ldots, M$), the ending time for the mth stream has to be at least $A + (A_m - A_{m-1})$. Hence the updated ending time for the mth stream from its previous value E_m right before the arrival of the new request is

$$E_m \leftarrow \max(E_m, A + A_m - A_{m-1}). \qquad (6)$$

We now find the stream requirement. Let $\overline{S}(R)$ be the average number of concurrent streams given R and W (Clearly, the pure-VOD case corresponds to $\overline{S}(0) = \lambda T_h$). We will consider the case where there are many servers and hence each new request is likely to come from a local server which is not currently in the multicast groups (i.e., number of servers $\gg \lambda W$). $\overline{S}(1)$ can be readily obtained by conditioning on the number of requests N eventually served by a basic stream. Given that $N = n + 1$, the first user holds the (basic) stream for a time T_h while the remaining n users have an arrival time which is uniformly distributed between 0 and W; hence the average total usage time of the streams is $T_h + nW/2$; the average total stream-time is then $T_h + \lambda W^2/2$. Since the time between successive stream allocation is given by $W + 1/\lambda$, the average number of concurrent streams is,

$$\begin{aligned} \overline{S}(1) &= \frac{1}{W + 1/\lambda}\left(T_h + \frac{\lambda W^2}{2}\right)\\ &= \frac{\lambda T_h}{1 + \lambda W}\left(1 + \frac{\lambda W^2}{2T_h}\right). && (7) \end{aligned}$$

(Note that the first term, $\lambda T_h/(1 + \lambda W)$ is the the average number of basic streams.) As mentioned before, $\overline{S}(1)$ first decreases and then increases with W. Therefore, W should not be larger than a value $\hat{W}$, which is obtained by setting $d\overline{S}(1)/d\hat{W} = 0$, i.e., $\lambda \hat{W}^2 + 2\hat{W} - 2T_h = 0$.

We have used simulation to study the case $R \geq 2$ due to its analytic difficulty. A lower bound on $\overline{S}(R)$, however, may be obtained by observing that $\overline{S}(R)$ achieves its minimum when $R \to \infty$. Furthermore, note that each new request requires a transient stream of length at least equal to the inter-arrival time between itself and its previous request (see Eqs. (4) and (6)). Given that there are $N = n$ requests latch on the basic stream (with $P(N = n) = p_n = (\lambda W)^n e^{-\lambda W}/n!$), the average inter-arrival time between adjacent request is $W/(n + 1)$; therefore,

$$\begin{aligned} \overline{S}(R) &\geq \overline{S}(\infty)\\ &= \overline{S}_b + \overline{S}_{trans}\\ &\geq \frac{T_h}{W + 1/\lambda} + \frac{1}{W + 1/\lambda}\left[\sum_{n=1}^{\infty} p_n \frac{nW}{n+1}\right]\\ &= \frac{\lambda T_h}{1 + \lambda W}\left[1 + \frac{W}{T_h}\left(1 - \frac{1 - e^{-\lambda W}}{\lambda W}\right)\right]. && (8) \end{aligned}$$

To obtain the storage used, note that the average number of concurrent requests is λT_h, each of which holds a buffer of size W minutes. Therefore $\overline{B} = \lambda T_h W$.

[2] In reality, the required buffer size would be slightly higher, due to the fact that the local server caches both the transient and basic streams (the buffer is filled at a higher rate) and the beginning portion of the movie has to stay in the buffer for a time W minutes before it can be deleted.

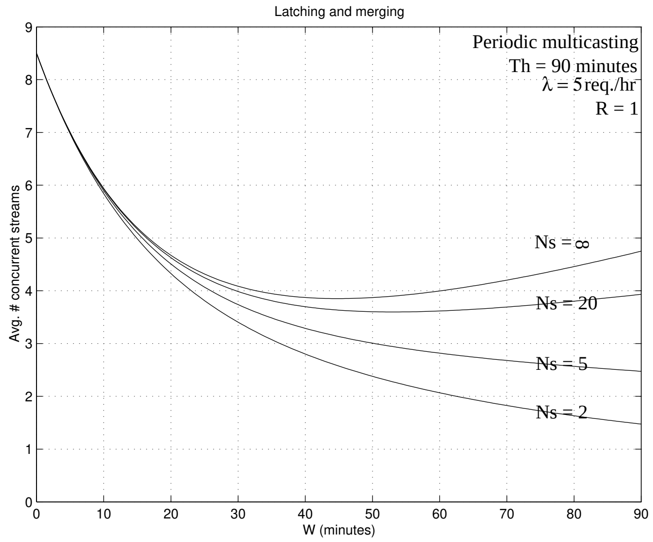

Fig. 3. $\overline{S}$ versus W given N_s.

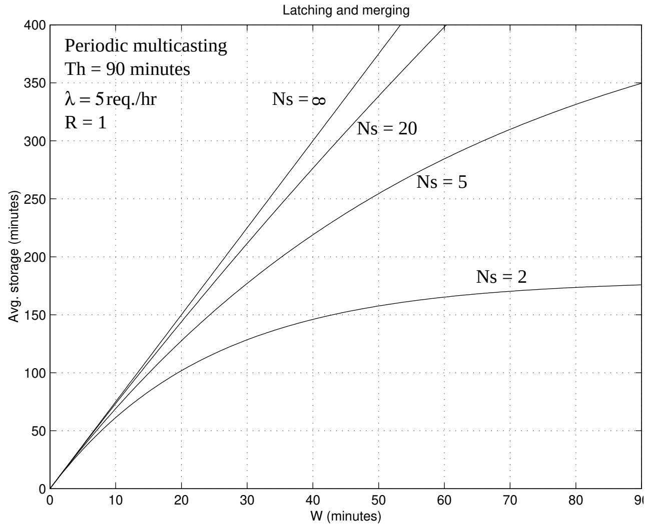

Fig. 4. $\overline{B}$ versus W given N_s.

V. Numerical Results

In this section, we provide some numerical examples for the periodic multicasting and on-demand multicasting.

A. Periodic multicasting

In this section, we study periodic multicasting with $R = 1$. We show in Fig. 3 $\overline{S}$ versus W given N_s ($\lambda = 5$ req./hr and $\lambda_i = \lambda/N_s$ for all i (uniform load)). As W increases, $\overline{S}$ first decreases and then increases (especially so for high N_s); therefore the system should be designed with W less than the point achieving the minimum number of streams. Below such a point, there is no much difference in $\overline{S}$ for different N_s. Most of the bandwidth saving is achieved at low W ($W \approx 10$–30 minutes). As N_s increases, it is more likely for an arrival to start a transient stream (instead of sharing a local cache) and hence $\overline{S}$ increases.

In Fig. 4 we show the average total storage $\overline{B}$ versus W given N_s, again with $\lambda_i = \lambda/N_s$ for all i ($\lambda = 5$ req./hr and $T_h = 90$ minutes). As W increases, it is more likely for the basic stream to continue and hence $\overline{B}$ approaches $N_s T_h$. Note that for large N_s, $\overline{B}$ increases quite linearly for a wide range of W (with the average storage goes as $(\lambda T_h)W$).

We show in Fig. 5 the trade-off between $\overline{S}$ and $\overline{B}$, given λ ($N_s = 10$ and $R = 1$). $\overline{S}$ first decreases with $\overline{B}$; however, as $\overline{B}$ increases further, the large W leads to longer transient streams and hence $\overline{S}$ increases. Suppose there is a cost associated with the storage and network streams, the figure can be used to find the optimal operating point achieving the lowest system cost by the following: We draw a line indicating the combination of $\overline{S}$ and $\overline{B}$ achieving the same total cost (the iso-cost line); with the line moving outwards as the cost increases. The point at which the cost line touches the trade-off curve minimizes the system cost. We clearly see that, with a linear cost function, the optimal point of the system is likely to be around the turning point in the trade-off graph.

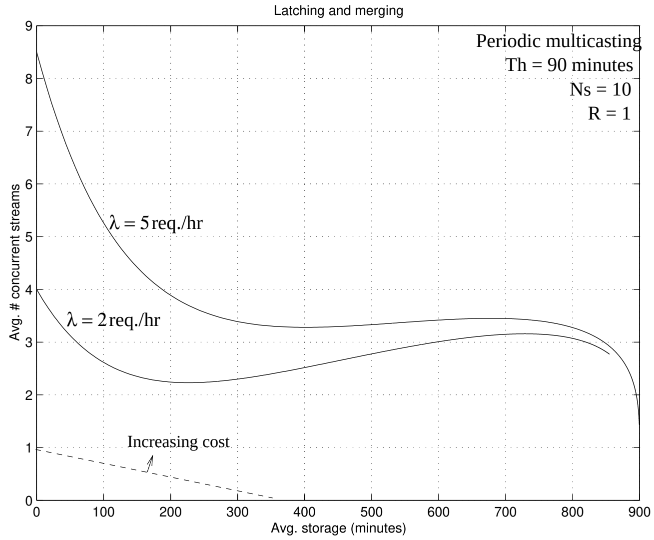

Fig. 5. $\overline{S}$ versus $\overline{B}$ given λ, for $N_s = 10$ and $R = 1$.

B. On-demand multicasting

We now study on-demand multicasting. We will study the case where N_s is large (essentially $N_s = \infty$), which is the pessimistic case for $\overline{S}$ and $\overline{B}$. From the periodic multicasting (Fig. 3), we also expect that such a case would not be much different from that with finite N_s.

We show in Fig. 6 $\overline{S}$ versus W, given R ($\lambda = 5$ req./hr, $T_h = 90$ minutes). As W increases, $\overline{S}$ decreases, first quite sharply and then very slowly, and then in some cases increases again. Note that the $\overline{S}$ corresponding to $R = 1$ differs from the case of the periodic multicasting (in Fig. 3 with $N_s = \infty$) by at most one stream (due to stream abortion in the periodic multicasting), showing the similarity between the two schemes. As R increases, $\overline{S}$ decreases (due to more latching and merging). There is, however, not much advantage in increasing R beyond 2 or 3.

We show in Fig. 7 $\hat{W}$ (the maximum window size at the local server) versus λ for $R = 1$ ($T_h = 90$ minutes). $\hat{W}$ first

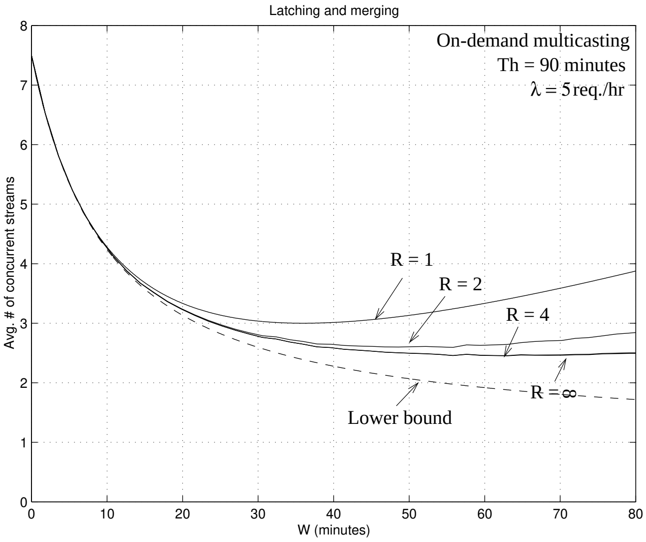

Fig. 6. $\overline{S}$ versus W given R ($\lambda = 5$ req./hr, $T_h = 90$ minutes).

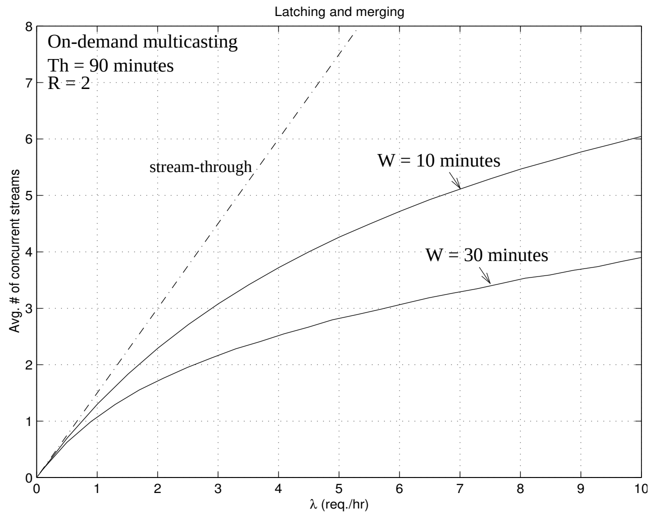

Fig. 8. $\overline{S}$ versus λ given W ($R = 2$, $T_h = 90$ minutes).

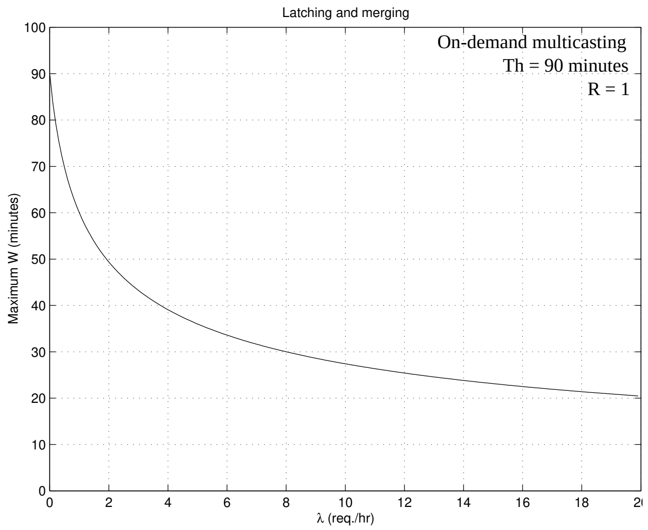

Fig. 7. $\hat{W}$ versus λ given T_h ($R = 1$).

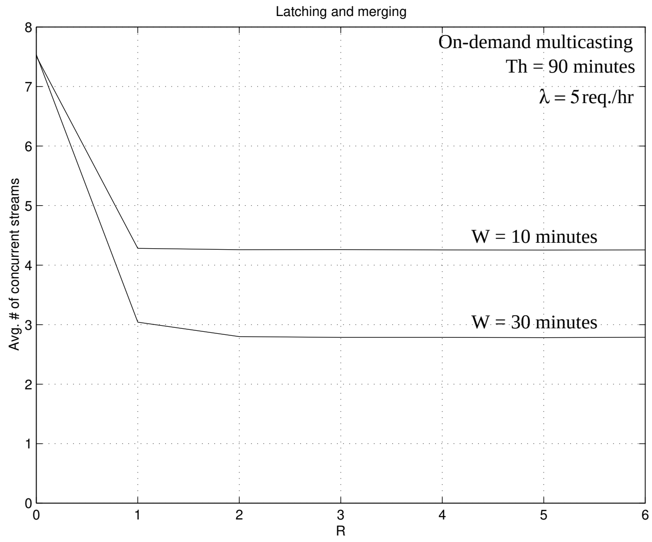

Fig. 9. $\overline{S}$ versus R given W.

decreases quite sharply with λ and then more slowly.

We show in Fig. 8 $\overline{S}$ versus λ given W, for $R = 2$. As λ increases, $\overline{S}$ increases, initially with slope T_h (as there is no latching and merging) and then more slowly. Also shown in broken line is the stream-through case (without multicasting and local caching). Compared with the stream-through case (i.e., there is no local caching and the repository delivers videos directly to the users), there is a substantial saving in $\overline{S}$ with latching and merging.

We show in Fig. 9 $\overline{S}$ versus R, given W ($\lambda = 5$ req./hr). There is a marked decrease in $\overline{S}$ when R increases from 0 (corresponding to the stream-through case) to 1; afterwards the decrease is (relatively) little. Therefore in reality, there is no need to latch on more than one or two streams.

Note that buffering would not be required if users can tolerate a certain delay. For example, if users can tolerate a delay of U minutes beyond which they renege, we can then group (i.e., batch) together the users arriving within a period of U minutes and serve them with a single multicast stream. If the first user in a batch always experiences a delay of U minutes, it can be shown that the average number of streams used is $\lambda T_h/(1 + \lambda U)$ (the auto-gated scheduling in [21], [22], [23]). Since in latching and merging all users within W minutes are also (eventually) served by a single multicast stream, it is worthwhile to compare the increase in $\overline{S}$ in this case with the request batching under the condition that $W = U$, so as to understand how many additional streams are necessary to decrease user delay. We show in Fig. 10 the average number of additional streams as compared with the batched case $\lambda T_h/(1 + \lambda W)$ so as to provide an on-demand service, given R ($\lambda = 5$ req./hr, $T_h = 90$ minutes). We see that if users can tolerate little delay, say $U = 10$ minutes, the additional cost in stream can be minor (only 0.2 additional stream over the original 4.1 streams); however, as user delay tolerance increases, say $U = 25$ minutes, such provisioning of an on-demand service takes on a higher stream cost (more than 30% increase in

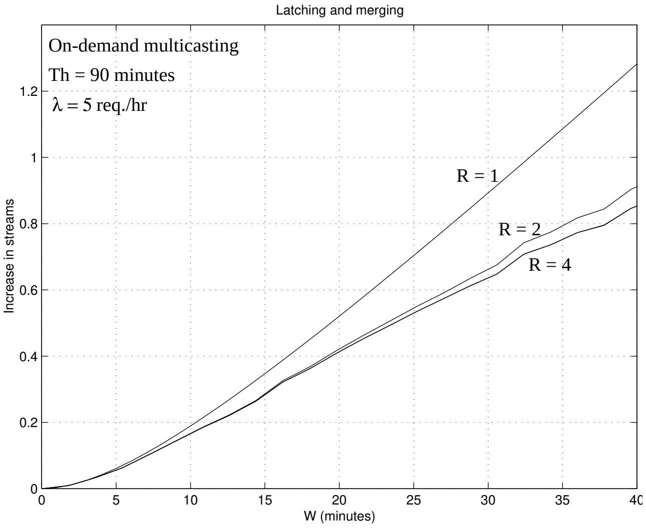

Fig. 10. Additional $\overline{S}$ required (as compared with the batched case $\lambda T_h/(1+\lambda W)$) versus W in order to offer on-demand services over a batching system, given R.

the streams over the original 2.43 streams).

VI. Conclusions

In a distributed servers architecture, a number of repository servers store all the video contents of interest to a large pool of geographically distributed users. Since the repository may have limited bandwidth and the transmission cost to the users may be high, a number of local servers can be placed close to the user clusters and cache the movies according to their local demands, thereby increasing the streaming capacity of the system and lowering the network transmission cost. In this paper, we have investigated such a system for on-demand video services using multicasting and local caching.

We have studied a scheme called "latching and merging." In the scheme, upon the arrival of a request the local server joins the existing multicasting stream for the movie and caches the data, while a transient stream is used to recover the time lag by supplying the missing (beginning) portion of the movie. After some time, the transient stream will then "merge" with the multicast stream and is relinquished; from then on, the movie can be served by the local cache and the multicast stream. Our caching scheme keeps a sliding window for a movie being displayed and hence the movie can be partially cached, thereby achieving better trade-off between network channels and local storage (and lower system cost) than treating a movie as a single entity. We have shown that our scheme is able to greatly lower the requirement in network channels as compared with a system without multicasting and local caching. Compared with request batching and multicasting, our scheme trades off buffer and stream costs for a lower user delay.

References

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

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

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

[4] K. Cleary, "Video on demand — competing technologies and services," in *Proceedings of International Broadcasting Convention*, (Amsterdam, Netherlands), pp. 432 – 437, 14–18 Sept. 1995.

[5] B. Furht, D. Kalra, F. L. Kitson, A. A. Rodriguez, and W. E. Wall, "Design issues for interactive television systems," *IEEE Computer Magazine*, pp. 25–38, May 1995.

[6] A. D. Gelman, H. Kobrinski, L. S. Smoot, S. B. Weinstein, M. Fortier, and D. Lemay, "A store-and-forward architecture for video-on-demand service," *Canadian Journal of Electrical and Computer Engineering*, vol. 18, pp. 37–40, January 1993.

[7] D. D. Harman, G. Huang, G.-H. Im, M.-H. Nguyen, J.-J. Werner, and M. K. Wong, "Local distribution for IMTV," *IEEE Multimedia Magazine*, pp. 14–23, Fall 1995.

[8] H. J. Stüttgen, "Network evolution and mutlimedia communication," *IEEE Multimedia Magazine*, pp. 42–59, Fall 1995.

[9] S. A. Barnett and G. J. Anido, "A cost comparison of distributed and centralized approaches to video-on-demand," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1173–1183, August 1996.

[10] S.-H. G. Chan and F. Tobagi, "Caching schemes for distributed video services," in *Proceedings of the 1999 IEEE International Conference on Communications (ICC'99)*, (Vancouver, Canada), June 1999.

[11] L. D. Giovanni, A. M. Langellotti, L. M. Patitucci, and L. Petrini, "Dimensioning of hierarchical storage for video on demand services," in *Proceedings of the IEEE ICC'94*, pp. 1739–1743, 1994.

[12] F. Schaffa and J.-P. Nussbaumer, "On bandwidth and storage tradeoffs in multimedia distribution networks," in *Proceedings of Infocom'95*, pp. 1020–1026, 1995.

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

[14] P. Lie, J. Lui, and L. Golubchik, "Threshold-based dynamic replication in large-scale video-on-demand systems," in *Proceedings of the Eighth International Workshop on Research Issues in Data Engineering*, (Orlando, FL, USA), pp. 52–59, 23-24 Feb. 1998.

[15] C. Papadimitriou, S. Ramanathan, P. V. Rangan, and S. SampathKumar, "Multimedia information caching for personalized video-on-demand," *Computer Communications*, vol. 18, pp. 204–216, March 1995.

[16] 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 *Proceedings of the IEEE Infocom'98*, (San Francisco, USA), pp. 660–667, March 29 - April 2 1998.

[17] H. Ghafir and H. Chadwick, "Multimedia servers – design and performance," in *Proceedings of Globecom*, pp. 886–889, 1994.

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

[19] G. H. Petit, D. Deloddere, and W. Verbiest, "Bandwidth resource optimization in video-on-demand network architectures," in *Proceedings of the 1st IEEE International Workshop on Community Networking*, (NY, New York), pp. 91–97, 13-14 July 1994.

[20] W. Liao and V. O. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, pp. 51–62, October-December 1997.

[21] S.-H. G. Chan and F. Tobagi, "On achieving profit in providing near video-on-demand services," in *Proceedings of the 1999 IEEE International Conference on Communications (ICC'99)*, (Vancouver, Canada), June 1999.

[22] S.-H. G. Chan, F. Tobagi, and T.-M. Ko, "Providing on-demand video services using request batching," in *Proceedings of the 1998 IEEE International Conference on Communications (ICC'98)*, (Atlanta, Georgia), pp. 1716–1722, June 7-11 1998.

[23] S.-H. G. Chan, F. Tobagi, and T.-M. Ko, "Bandwidth planning in near video-on-demand," in *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems (ISCAS'98)*, (Monterey, CA), pp. 61–64, May 31 - June 3 1998.