Caching Schemes for Distributed Video Services

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

Abstract— **In an on-demand video system, the repository servers storing all the video contents generally have limited streaming capacities and may not be co-located with the users. To achieve higher user capacity and lower network transmission cost, distributed servers architecture can be used. In such a system, multiple local servers are placed close to the user pools and, according to their local demands, the servers dynamically cache the contents streamed from the repository. We study in this paper a number of caching schemes by considering whether the local servers can exchange their cached contents among themselves or not. All the caching schemes keep a sliding window worth of data for each video being displayed; hence a video can be partially stored. We study the trade-off between the storage capacity required and the network channel used in each scheme, and address how the system cost can be minimized by appropriately sizing the window. We also show the cost advantage in using such a system over a system of request batching and multicasting in providing distributed video services.**

Keywords— **Video services, distributed servers architecture, caching schemes, network channel cost, storage cost**

I. Introduction

Advances in computer and networking technologies have made on-demand video services (video-on-demand (VOD)) a reality. There are important video applications pertaining to entertainment, education, advertising and information, such as movie-on-demand, distance learning, home shopping, interactive news, etc.[1], [2], [3], [4]

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, 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 and stream 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 taking 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], [5]. By putting more repository servers and local servers, the system offers scalable storage and streaming capacities to provide distributed video services.

We consider 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. Suppose that a 1 GB of disk (costing today about \$200) is in use for a certain number of hours per day, say 6 hours, is to be amortized over a period of one year. The cost of storage

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.

TABLE I

Values of β and γ (based on 1 GB disk of \$200, amortized over a year with 6 hours/day usage; $b_0 = 5$ Mb/s).

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

is then $\$200/(365 \times 6) = \$0.09/(\text{GB}\cdot\text{hr.})$. Consider the streaming rate of a movie to be $b_0 = 5$ Mb/s (MPEG-II quality); then one minute worth of video data costs $\$3.42\times10^{-3}$ per minute of storage. We also consider that a cost is associated with the streaming of a movie from the repository over a network channel to the remote users. The cost may range from low (e.g., when a network like the Internet is used) to quite high (e.g., when satellite channels are used). In Table I, we show the cost of using a channel per minute, β (With $\beta = \$0.03/(\text{minute}\cdot\text{channel})$, the delivery of a 100-minute movie would cost \$3, a reasonable figure for on-demand services). We also show a parameter γ, given by the ratio between the storage cost given above (i.e., $\$3.42\times10^{-3}$/minute to store a 1-minute video clip) and β, and we see that γ is likely to be in the range from $10^{-4} - 10^{-2}$ for on-demand services.

There is a trade-off between using a long-haul network channel to deliver a movie and storing a 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 so as not to incur too much local storage cost.

Indeed, some movies are popular while others are not, and the popularity may change over time. There is hence a need to decide which movie and how much of the movies to store locally given movies' request rates. Since the request rate of a movie changes over time, it is important that such a decision be made continuously over time to take into consideration the dynamic nature of the request rate. (In other words, the repository and local servers may have to be constantly exchanging movies.) As an example, consider a geometric video popularity model in which the access probability of the movies follows a geometric distribution [6], [7]. We show in Fig. 1 the request rate for movie i in a system with 500 movies and the total request rate $\Lambda = 4000$ req./hr (the notation r/m represents that r% of the requests ask for m% of the movies). We see that the request rate of a movie can differ quite widely 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., un-cooperating service providers or security issues). In this way, the local servers obtain their movies only from the repository and operate independently from each other. We show in Fig. 2 an example of such a case for a cable TV sys-

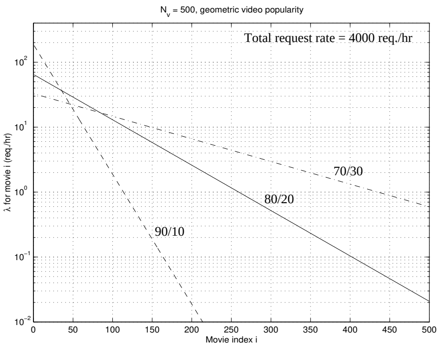

Fig. 1. λ for movie i in a video system with 500 movies.

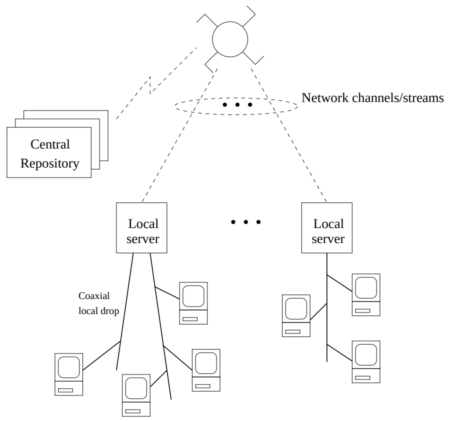

Fig. 2. A system for distributed video services.

tem. Multiple head-end servers serve their local communities through coaxial local drops. The central repository delivers the movies with the use of network channels (via some considered to be low-cost channels such as the Internet or relatively high-cost satellite channels). The channels may be leased by the video content service providers and always available, or they may be acquired on-demand with some cost.

If a service provider operates a number of local servers, the servers can be connected by a network so that they can communicate with each other and obtain the missing data from the other nearby servers (instead of from the repository). This is possible for servers co-located in, for example, a campus network, an entertainment network with cooperative servers/service providers, a private enterprise network with low transmission cost, etc. By making a local copy of the streamed data, a given server can serve the subsequent local request(s) by itself, hence decreasing the network bandwidth requirement. The repository may multicast movies to the local servers, or it may unicast the movie to a given local server, which in turn multicasts the movie to the other servers using their interconnected network.

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 paper, we examine the conditions under which a movie should be stored in order to minimize the total cost, given cost functions of storage and network channels. We consider the cases of independent servers and communicating servers, with the repository using unicast network channels in delivering movies (the use of multicast channels has been discussed in [8], [9]).

We study here a number of caching schemes in the local servers. The caching schemes keep a sliding window for each movie being streamed and hence is able to keep a portion of the movie locally (i.e., *partial* caching). All requests arriving before the beginning portion of the stored movie is deleted can be served directly from the local cache (known as a *cache group*). (Keeping a window also offers a certain degree of interactivity with the movie.) Since movies can be cached partially, the channel cost and storage cost can be optimally traded off with each other. We consider that the network channels can be acquired on demand, and the latency in the central repository is low and can be ignored.

We briefly discuss previous work as follows. Dan et al. consider that the segments of a movie file can be dynamically replicated depending on the load [10]. Though partial file storage is considered, network issues (bandwidth requirement and cost) have not been addressed. Wang et al. consider storing part of the "bursty" movie frames in local servers, so that the requirement in network bandwidth can be reduced [11]. Their work assume knowing the bandwidth profile of each movie, while our work does not need such an assumption. Barnett and Anido compare the set-up cost of a centralized video system with that of a distributed system [5], while we study here the *running* cost of a video system, and with different cost functions. Giovanni et al. consider a hierarchical storage system in which local servers get their videos from a repository through some fixed and leased network channels [6], while our work discusses on-demand channels. Schaffa and Nussbaumer study a VOD system in which the servers are arranged as a hierarchical balanced tree, and all video files are replicated at a certain level in the tree [12]. Papadimitriou et al. consider that a repository is given a prespecified viewing schedule and has to deliver the videos with minimum cost [13]. Little and Venkatesh consider a system in which video files can be replicated in the servers in order to satisfy user requests [14]. Lie et al. study a distributed VOD system in which movies are dynamically replicated and deleted to achieve load balancing among the servers [15].

Most of the above work treat a video file as a single entity, and hence is either completely stored or not at all. We consider here partial caching, and hence achieving better trade-off. We have also considered on-demand network channels with a cost associated with using them, which has not been discussed above.

This paper is organized as follows. We first describe in Sect. II the caching schemes we study in this paper. In Sect. III, we analyze the schemes in terms of their storage and channel requirements, and address how cost can be minimized. In Sect. IV we provide some numerical results, and in Sect. V we compare the cost advantage of a distributed servers architecture over a batching system. We conclude in Sect. VI.

II. Caching Schemes

A. Independent servers

Since the local servers are operated independently, we can focus on an arbitrary pair of the repository and a local server. We discuss the following caching schemes depending on whether the size of the sliding window is fixed or is driven by the arrivals:

• Fixed window size — Here the window size is fixed. To understand the caching scheme, let's first consider that initially no user is being served in a local server and no movie is cached locally. An arrival at the server leads to an allocation of a network channel from the repository to stream the movie, while the local server, in anticipation of its future requests, caches the movie data for a period of time equal to W minutes before the data is deleted (the window size is hence the data lifetime). Therefore, all requests including the first one arriving within W minutes after the allocation of a network channel form a cache

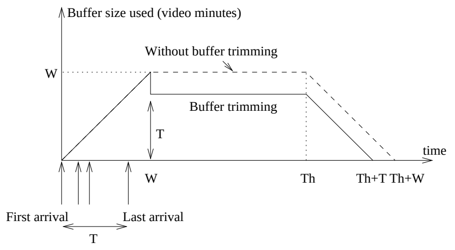

Fig. 3. Caching using fixed window size with or without buffer trimming.

group.

In above, a sliding window of W minutes is kept for a movie after it is brought into the local cache. Obviously, buffer requirement can be reduced if, after we have collected a cache group, we "trim the buffer" and keep only the amount of data just necessary to serve the first and the last arrival in the group. We show in Fig. 3 the buffer requirement for caching using fixed window size with or without such trimming, given that the interval between the first and the last request in the cache group is T minutes. With trimming, the buffer is cut to T video minutes after W minutes.

• Request-driven window size — With the fixed window size, an arrival more than W minutes after the first request in a cache group will need a new network stream. To take into account of temporal locality, we can dynamically adjust the window size according to the arrival process: Data brought into a local server has an initial window size of W minutes; if a new request arrives within this time, the window size is extended from that point on to a further of W minutes (i.e., the window size is extended by the interarrival time of the request from its previous one).

B. Communicating servers

We consider that data is copied locally from a remote streaming server so that future requests for the movie can be served locally. Our scheme is the same as the "chaining" mentioned in [16], in which the saving in bandwidth has been studied using simulation. We quantify such saving in this paper using analytic method.

In this scheme, each local server uses fixed or request-driven window size. If before the beginning portion of a movie is deleted in a local cache and another request of the same movie arrives at a remote server, the movie will be copied to the remote server with an initial lifetime of W minutes. Therefore, movie data can be relayed from one server to another, and hence the servers are "chained." A chain is "broken" when two successive requests are separated by more than W minutes, in which case a new network stream has to be allocated in the repository to serve the later arrival.

III. Analysis

We study the storage and channel requirement for each caching scheme. Since network channels can be acquired on-demand, the servicing of requests pertaining to a given movie is independent of the servicing of requests for other movies, and hence it suffices to consider the single movie case. In the following, we consider a movie of length T_h minutes, with its request process being Poisson with rate λ req./min. and its streaming rate being b_0 Mb/s. We are interested in the following:

• The average number of channels (or concurrent streams) $\overline{S}$ used for the movie;

• The average buffer requirement for the movie in a local server $\overline{B}$ (in minutes of video time); and

• The total system cost comprising the channel cost and local storage cost (the streaming cost from the local servers to their users is considered negligible). Note that the storage or channel cost may be any arbitrary function. For illustration purpose, we consider linear cost functions, though our study is not limited to it. Let β be the cost per minute in using a network channel (in \$/(minute·channel)), and α be the storage cost per minute for each MB used (\$/(minute·MB)). The average cost per minute is then $\alpha(60/8b_0)\overline{B} + \beta\overline{S}$. Defining $\hat{C}$ as the normalized cost with respect to β and recalling that $\gamma \triangleq (60/8b_0)\alpha/\beta$, we have $\hat{C} = \gamma\overline{B} + \overline{S}$.

For communicating servers, the total system cost is the sum of the storage cost in all the local servers and the network channel cost (the communication cost between local servers is considered negligible). Given that there are N_s local servers each with buffer requirement $\overline{B}_i$ $(i = 1, 2, \ldots, N_s)$, the total normalized cost is $\hat{C} = \gamma\sum_{i=1}^{N_s}\overline{B}_i + \overline{S}$. Given $\hat{C}$, the average cost per user is given by $\beta\hat{C}/\lambda$.

A. Independent servers

We first consider caching with fixed window size without trimming. The average interval between successive channel allocations is given by $W + 1/\lambda$, and hence the average number of channels is $\overline{S} = T_h/(W + 1/\lambda)$. Since each bit of the movie file is held in the local server for W minutes, the average buffer size is $\overline{B} = T_hW/(W + 1/\lambda) = \lambda WT_h/(1 + \lambda W)$. We see that $\overline{S} = \lambda T_h - \lambda\overline{B}$, and hence $\hat{C} = \gamma\overline{B} + \overline{S} = (\gamma - \lambda)\overline{B} + \lambda T_h$. Therefore, for linear cost functions, minimizing $\hat{C}$ means that we either don't cache at all (the case corresponding to $\gamma > \lambda$, making $\overline{B} = 0$) or store the whole movie in the local server (the case corresponding to $\gamma \leq \lambda$, making $W = \infty$ and hence $\overline{B} = T_h$).

We now consider buffer trimming. First note that $\overline{S}$ is the same as the un-trimmed case. We now obtain $\overline{B}$. Given that the interval between the first and the last request in a cache group is T minutes (where $\leq W \leq T_h$), by referring to Fig. 3 again, the area under the graph is given by $A = W^2/2 + (T_h - W)T + T^2/2$. Hence, $E[A] = W^2/2 + (T_h - W)E[T] + E[T^2]/2$. By conditioning on the number of arrivals $n \geq 0$ within the time W (excluding the first request in the cache group), we have $E[T|n] = nW/(n+1)$ and $E[T^2|n] = nW^2/(n+2)$. Removing the condition on n (which is Poisson with mean λW) and noting the time between successive network stream allocations as $W + 1/\lambda$, we then obtain $\overline{B}$. The case corresponding to $W \geq T_h$ can be obtained similarly with $E[A] = T_h^2/2 + (W - T_h)T_h + E[T^2]/2$.

We next consider the request-driven window size. An arrival will extend the window size if its arrival time is within W minutes from the previous request, i.e., with probability $p = 1 - e^{-\lambda W}$. Therefore, the expected inter-arrival time given that successive requests are within W is $T_a = (1/p)\int_0^W \lambda x e^{-\lambda x}dx = 1/\lambda - W/(e^{\lambda W} - 1)$. The average lifetime data stays in the local server is hence $L = \sum_{i=0}^{\infty} p^i(1-p)iT_a + W = W + T_a e^{\lambda W}$. The average interval between successive channel allocation is hence $T_s = L + 1/\lambda$; and the average number of concurrent streams is $\overline{S} = T_h/T_s$. The average amount of buffer used for a user is given by $T_ap + W(1-p) = (1 - e^{-\lambda W})/\lambda$. Since there are on average λT_h concurrent users, $\overline{B} = T_h(1 - e^{-\lambda W})$.

B. Communicating servers

Let N_s be the number of communicating servers. We consider that N_s is reasonably large so that a request is likely from a server different from the servers already chained. The expression for the average number of concurrent streams is the same as that

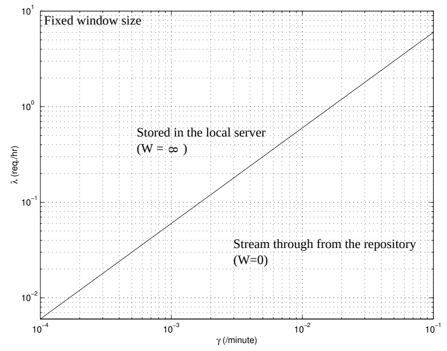

Fig. 4. Relationship between λ and γ to minimize cost.

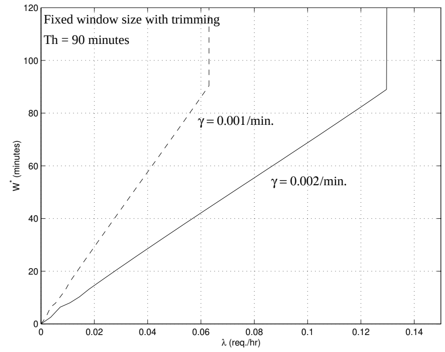

Fig. 5. W^* versus λ given γ.

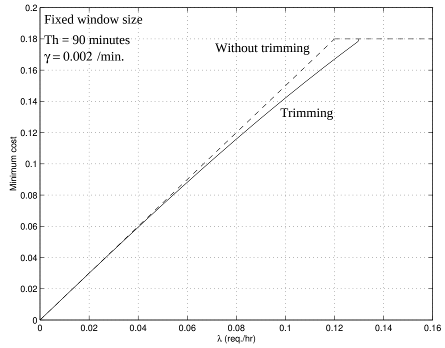

Fig. 6. $\hat{C}^*$ versus λ, with or without buffer trimming.

of the case of request-driven window size. The average buffer requirement in the system is given by $\overline{B} = \lambda T_h W$.

IV. Numerical Examples

In this section, we will study the trade-off between $\overline{S}$ and $\overline{B}$ for each caching scheme, and address how much data should be cached in order to minimize the system cost. (Note that λ is shown in req./hr in all our plots.)

A. Independent servers

We first study the fixed window size without trimming. Recall that the trade-off between $\overline{S}$ and $\overline{B}$ given λ is linear with slope $-\lambda$, and with linear cost functions the optimal caching strategy to minimize the total system cost 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. This is shown in Fig. 4. We see that when γ is low (storage is cheap relative to channel cost) or λ is high (popular title), the entire movie is likely to be stored locally.

We now study the effect of buffer trimming. We show in Fig. 5 the W^* required to minimize cost versus λ, given γ. As λ increases, W^* increases quite linearly until a point beyond which $W^* = \infty$ (storing the movie completely). While in the un-trimmed case W^* is either 0 or ∞, in the trimmed case there is a non-zero finite W^* to achieve minimum cost, i.e., partial movie caching. As λ increases, the buffer requirement to achieve minimum cost also increases, until the point when the movie is completely stored. We remark here that $\overline{S}$ increases with λ, and then drops to zero when $\overline{B} = T_h$. (For a given λ, as γ decreases, W^* increases.) We show in Fig. 6 the minimum cost $\hat{C}^*$ versus λ, with or without buffer trimming ($\gamma = 0.002$ and $T_h = 90$ minutes). Despite the fact that trimming achieves lower buffer requirement, due to the low storage cost, the cost advantage does not turn out to be significant, with the maximum difference at $\lambda = \gamma$. The saving in trimming increases with γ.

We now consider caching with request-driven window size. We show in Fig. 7 the trade-off between $\overline{S}$ and $\overline{B}$ given λ. The line is quite linear and resembles that of the fixed window size (with slope $\approx -\lambda$). Also shown is a "cost line" indicating the combination of $\overline{S}$ and $\overline{B}$ to achieve the same total system cost, with the line moving outwards as cost increases. The optimal operating point is the point at which the cost line touches the trade-off curve. With arbitrary stream and storage cost functions, the cost line may touch any point on the trade-off curve. However, with linear cost function, the movie is either streamed through the local server from the repository or completely cached depending on if $\lambda < \gamma$ or not (as in the case of the fixed window size).

B. Communicating servers

We now consider communicating servers. We plot in Fig. 8 W^* versus λ ($\gamma = 0.002$/min.). We see that when λ is too low, the movie should be directly streamed from the repository (no caching). As λ increases, W^* first increases very sharply and then decreases more slowly.

We show in Fig. 9 $\hat{C}^*$ versus λ for chaining (with $W = W^*$ at each λ). As λ increases, $\hat{C}^*$ also increases, first quite fast and then more slowly. Compared with the stream-through case, chaining achieves substantial cost saving.

We finally show in Fig. 10 W^* versus γ, given λ. As γ increases, W^* decreases (first rather exponentially and then more slowly). There is a value of γ beyond which the storage cost relative to the channel cost is so high that streaming-through becomes more cost-effective.

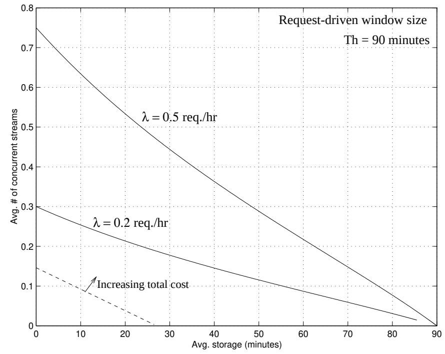

Fig. 7. $\overline{S}$ versus $\overline{B}$ given λ for request-driven window size.

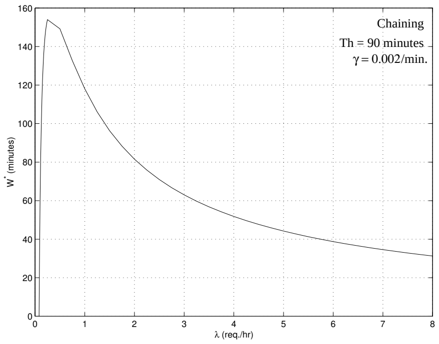

Fig. 8. W^* versus λ for chaining.

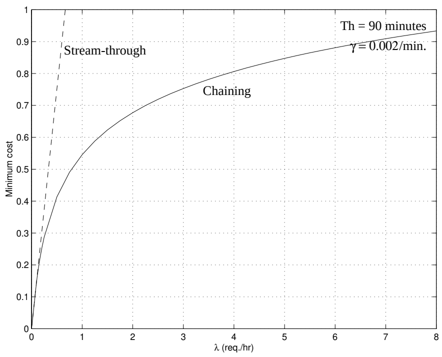

Fig. 9. $\hat{C}^*$ versus λ.

V. Cost Comparison With a System Using Request Batching and Multicasting

Consider a system in which the repository is used to serve all the requests. To increase the user capacity, request batching and multicasting is used (Ref. [2], [17], [18], [19] and references therein). If users can tolerate a maximum delay of D_{max} minutes, the batching period can be set to 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. It can be shown that the average number of streams required, and hence the normalized total system cost, is $\overline{S} = \hat{C} = \lambda T_h/(1 + \lambda D_{max})$ [17].

A distributed servers architecture achieves virtually no user delay by means of streaming servers. This advantage, however, comes with the cost of additional servers and storage. However, as compared with request batching, it is important to note that

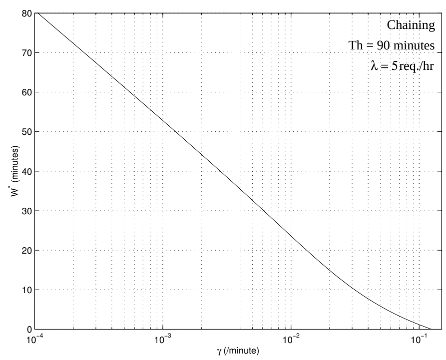

Fig. 10. W^* as a function of γ given λ for chaining.

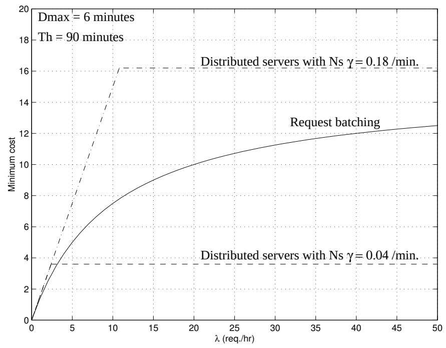

Fig. 11. $\hat{C}^*$ versus λ.

depending on the acceptable user delay, it is possible that the total cost of a distributed servers architecture be yet lower.

Let's consider a distributed servers architecture using caching with fixed window size. Let N_s be the number of servers in the system, with the request rate at server i for a movie being λ_i. Obviously, $\sum_{i=1}^{N_s} \lambda_i = \lambda$. The minimum cost is hence $\hat{C}^* = \sum_{i=1}^{N_s} [\gamma T_h u(\lambda_i - \gamma) + (1 - u(\lambda_i - \gamma))\lambda_i T_h]$, where $u(x)$ is the unit-step function with $u(x) = 1$ for $x \geq 0$ and $u(x) = 0$ otherwise.

We show in Fig. 11 the cost of the distributed servers architecture and the system of request batching and multicasting versus λ given γ, with $D_{max} = 6$ minutes and $\lambda_i = \lambda/N_s$ (we have considered that the cost of unicast channels is the same as that of multicast channels). For values of $N_s\gamma$ higher than a certain value ($= 1/D_{max}$), the distributed servers architecture has a higher cost than the batching system, hence trading higher system cost for lower user delay. However, with lower $N_s\gamma$, the distributed servers architecture can achieve lower cost than request batching, while at the same time offer lower delay to the users. Their cost lines cross at $\lambda' = N_s\gamma/(1 - N_s\gamma D_{max})$. To minimize the system cost, for $\lambda < \lambda'$ we use request batching to deliver movies from the repository to the users, and for $\lambda > \lambda'$ we store the movie at the local server.

As an example to show the cost saving of a distributed servers architecture, let's consider a system of 80/20 movie popularity with total arrival rate Λ req./hr (ref. Fig. 1), using the parameters corresponding to the column $\beta = 0.03$/minute in Tab. I. We compare the cost of the following seven systems: 1) *VOD*: the repository serves all the requests on the demand basis; 2) *All stored locally*: a system with N_s local servers only, each of which replicates all the movies and serves Λ/N_s req./hr; 3) *batching*: the repository uses request batching and multicasting to serve all the requests; 4) *20% local movies*: a system with a repository and N_s uniformly-loaded local servers each storing the top 20% of the most popular movies; 5) *optimal caching*: a distributed servers architecture with a repository and N_s uniformly-loaded local servers using fixed window size to achieve minimum cost; 6) *combined scheme*: a system of a repository and N_s uniformly-loaded local servers, with movies stored in the local servers if $\lambda \geq \lambda'$, and batching and multicasting in the repository if $\lambda < \lambda'$; and 7) *chaining*: a chaining system with many local servers.

We show in Fig. 12 the cost C versus the total request rate Λ, for the seven systems ($N_s = 20$ and $D_{max} = 6$ minutes). From the cases of VOD and storing all the movies locally, we see that when Λ is low, we should use VOD, and when Λ is high, we should store all the movies locally. Batching can reduce the cost substantially compared with the VOD case, but

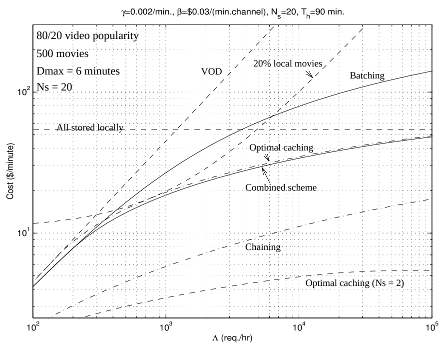

Fig. 12. The system cost per minute versus Λ.

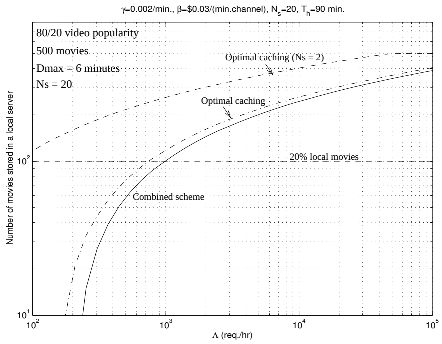

Fig. 13. Optimal number of movies stored locally versus Λ.

when Λ is high, its cost can still be higher than that of storing the movies locally. Storing 20% of the most popular movies locally achieves good cost saving for a certain range of Λ, showing that naively storing 20% most popular movies in a local servers does not always achieve good cost saving. The optimal caching and combined scheme achieve low cost, though their costs are similar (as batching is not effective for the non-popular movies). Chaining achieves the greatest saving in cost. As N_s decreases, the cost corresponding to the optimal caching decreases (due to lower data duplication). We see that by taking advantage of today's low storage cost, optimal caching can achieve great saving as compared with a batching system. Even greater cost advantage can be achieved if video popularity is more skewed.

We show in Fig. 13 the optimal number of local movies versus Λ. As Λ increases, the number of local movies also increases (first quite sharply and then very slowly) and approaches the total number of movies in the system. Again we see that there is not much difference between the optimal caching and the combined scheme.

VI. Conclusions

Distributed servers architecture is able to achieve storage and streaming scalability. In such a system, a number of repository servers store all the video contents of interest to a large number of geographically distributed users. Due to limited repository capacities and the likely high transmission cost from the repository to the users, using only the repository to serve the users would not be cost-effective. To solve the problem, local servers can be put close to user clusters and cache the video contents according to their local demands. We have studied a number of caching schemes for providing such distributed video services depending on whether the local servers can exchange video contents with each other or not. We have studied the trade-off between the number of network channels required and the local storage capacity. Given certain cost functions, we have addressed when (i.e., under what conditions) and how much to cache in order to minimize the system cost.

Our caching schemes are effective in reducing the network bandwidth by caching a movie partially. Generally, the unpopular movies should be streamed directly from the repository, and the popular ones should be stored locally. Movies of intermediate popularity are likely to be partially stored in the local servers. If the local servers can exchange data with each other, the system cost can be greatly reduced. Given the low storage cost today, distributed servers architecture effectively trades off storage cost with communication cost. The more skewed video popularity is, the lower the cost is. As compared with a system using request batching and multicasting, distributed servers architecture can achieve both lower overall system cost *and* 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] 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.

[5] 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.

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

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

[8] S.-H. G. Chan and F. Tobagi, "Scalable services for video-on-demand." Technical report, CSL-98-773, Computer Systems Laboratory, Stanford University, December 1998.

[9] S.-H. G. Chan and F. Tobagi, "Providing Distributed on-demand video services using multicasting and local caching," in *Proceedings of IEEE Multimedia Applications, Services, and Technologies*, (Vancouver, Canada), 7 June 1999.

[10] 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.

[11] 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.

[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] 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.

[14] 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.

[15] 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, NR, 23-24 Feb. 1998.

[16] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, (Ottawa, Ont., Canada), pp. 110–117, IEEE, 3-6 June 1997.

[17] 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.

[18] 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.

[19] S.-H. G. Chan and F. Tobagi, "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.