

ON SCALABILITY OF CLOSED-LOOP VIDEO DELIVERY SERVICE

Min-You Wu and Wei Shu

Department of Electrical and Computer Engineering
The University of New Mexico

ABSTRACT

Scalability is a key issue of on-line video service. Different methods have been proposed for the closed-loop video service, such as batching and patching. However, scalability itself has not been formally defined and the condition for a scalable system is not well studied yet. This paper studies scalability and the condition for scalable systems. It is found that there is a turning point for a scalable system, that is, a minimal request arrival rate which depends only on the number of videos, video length, and request distribution. The turning points for batching and patching are presented. The scalability of a new method, scheduled video delivery (SVD), which is able to improve scalability of closed-loop video service, is also studied in this paper.

1. INTRODUCTION

Video-on-Demand (VoD) is gaining popularity in recent years with the proliferation of broadband networks. True VoD is still expensive since the server and the network deliver videos for individual requests. It is not scalable because of its high bandwidth requirement.

It is critical to investigate new methods for scalable video delivery. Two approaches for scalable video delivery have been developed. The open-loop approach [1, 2, 3, 4, 5] requires no return path so it can be used for one-way cable systems, whereas the closed-loop approach [6, 7, 8, 9, 10, 11] requires a two-way system. The open-loop system continually broadcasts a video even if no one watches it. The closed-loop system only delivers the requested videos to users, thus is more efficient.

In the closed-loop approach, a number of methods, such as batching [6, 7, 8, 12, 13] and patching [14, 10, 15] have been proposed to combine more requests to

minimize the number of broadcast or multicast streams. Other methods, which are similar to patching, include catching [16], stream tapping [17], and stream merging [18].

Patching has been claimed as a scalable method. However, this is not always true since many requests cannot be combined. In fact, patching is not scalable when the arrival rate of requests is less than a certain level. It is true also for the batching method. The scalability issue has not been well studied. Furthermore, the condition of a scalable video system is unknown. The relation between scalability and system parameters such as the number of videos, the video length, request distribution, and request arrival rate is not known either. The topic of this paper is to study effects of these parameters. The major finding is that there is a minimal request arrival rate for each method. Analysis shows that only after this turning point a system becomes scalable where the number of streams required does not increase or slowly increases with the number of requests. In this paper, the analysis for batching and patching is presented. The analysis shows that a system with a large number of videos is less scalable. A video repository of more than 1,000 video objects results in a non-scalable system in practical situation. A new method, scheduled video delivery (SVD), is able to improve scalability. The analysis of SVD is also presented in this paper.

2. PARAMETERS IN A VIDEO DELIVERY SYSTEM

There are two components in a video delivery system, the stored videos and the user requests. Two parameters describe the property of videos:

- the number of videos in the video repository (N)

- the length of videos (L)

We assume all videos have the same length in this paper. Another parameter is whether videos are constant-bit-rate (CBR) or variable-bit-rate (VBR), which has little influence to scalability and therefore is not considered in this study.

Two parameters describe the property of user requests:

- request distribution
- request arrival rate (a)

The probability that a request accesses video j is

$$p_i = \frac{C}{i},$$

where

$$C = \frac{1}{\sum_{i=1}^N \frac{1}{i}}.$$

This is called the Zipf distribution [19]. Video 1 is the most popular and video N is the least popular. The Zipf distribution can be generalized as follows:

$$p_i = \frac{C}{i^\alpha},$$

where

$$C = \frac{1}{\sum_{i=1}^N \frac{1}{i^\alpha}}.$$

Most of this study assumes $\alpha = 1$. A sensitivity analysis of α will be given at the end of this paper. For simplicity, we assume that the request arrival times are evenly distributed.

Using these four parameters to model a video delivery system, analysis is performed for various delivery methods in the following sections.

3. SCALABILITY OF BATCHING

A simple VoD service serves each request individually. That is, a video stream is issued for each request and the number of video streams required is aL , where a is the request arrival rate and L is the length of videos. Thus, a simple VoD service is not scalable. With broadcast or multicast, the requests arrived at the same time can be combined and served together. For example, assume the video length is L seconds and all

requests arrived in one second are served by a single stream, the maximum number of simultaneous streams is NL , where N is the number of videos and L is the length of videos.

Batching combines the requests arrived in some time period T , say, 600 seconds, and serves them together by one stream. It is a simple method, but the requests that arrived earlier must wait for at most T time and the average waiting time is $T/2$. At most NL/T streams are required, independent of the number of requests.

When the request arrival rate is a , the arrival rate for video i can be computed by

$$\lambda_i = ap_i = \frac{Ca}{i}.$$

Assume the batching time is T , then $\lambda_i T$ requests can be combined. When $\lambda_i T < 1$, no request can be combined. Since λ_i is monotonously decreased with i , the entire video repository can be divided into two sets by a *turning point* v . One set is from video 1 to video v where requests can be combined and the other set is from video $v + 1$ to video N where requests cannot be combined.

The value of v can be obtained by

$$v = \text{Max}(i) | \lambda_i T > 1.$$

Without lossing generality, assume $\lambda_v T = 1$. Thus,

$$\lambda_v T = \frac{C}{v} a T = 1,$$

and

$$v = CaT.$$

Because the upper limit of v is N , we have

$$v = \text{Min}(CaT, N).$$

Let $r = \sum_{i=1}^v \frac{C}{i}$, which is the percentage of requests on the first set of video. The number of requests on the first set of videos is raL . Since $r = \sum_{i=1}^v \frac{C}{i} = \sum_{i=1}^v \frac{1}{i} \times C = \sum_{i=1}^v \frac{1}{i} / \sum_{i=1}^N \frac{1}{i}$, $r \leq 1$. Also, since $C/r = \frac{1}{\sum_{i=1}^v \frac{1}{i}}$, $C \leq r$. Then we have

$$C \leq r \leq 1.$$

The number of streams required for the first set of videos is $v \frac{L}{T}$ as at most $\frac{L}{T}$ streams are required for each video.

The requests can not be combined for the second set of videos. Thus, the number of streams is equal to the number of requests on this set of videos:

$$(1 - r)aL.$$

Thus, the number of streams for request arrival rate a is

$$m = v\frac{L}{T} + (1 - r)aL.$$

When $CaT < N$,

$$m = CaT\frac{L}{T} + (1 - r)aL = (C + 1 - r)aL.$$

Since $r \leq 1$, $m = (C + 1 - r)aL \geq CaL$. The number of streams required for the simple VoD is aL and C is a constant for a given value of N . Thus, the number of streams required is proportional to the number of requests and the system is not scalable under this condition.

When $CaT \geq N$, since $r = 1$ under this condition,

$$m = N\frac{L}{T}.$$

The number of streams required is independent of the number of requests. The system is scalable. For a given number of videos, N and C are constant. Thus, when $a < \frac{N}{CT}$, the system is not scalable and when $a \geq \frac{N}{CT}$, the system is scalable.

We can define a parameter, the *reduction rate (RR)*, to measure the improvement of batching to the simple VoD:

$$RR = \frac{aL}{m}.$$

For batching,

$$RR = \frac{aL}{v\frac{L}{T} + (1 - r)aL}.$$

When $CaT < N$,

$$RR = \frac{aL}{(C + 1 - r)aL} = \frac{1}{C + 1 - r} < \frac{1}{C}.$$

Since C is a constant which is normally larger than 0.1, RR is normally less than 10.

When $CaT \geq N$,

$$RR = \frac{aL}{\frac{NL}{T}} = \frac{aT}{N}.$$

Here, RR increases with a and the system is scalable.

When α is not equal to 1, the value of v and r should be changed to:

$$v = \text{Min}((CaT)^{\frac{1}{\alpha}}, N)$$

and

$$r = \sum_{i=1}^v \frac{C}{i^{\alpha}}.$$

4. SCALABILITY OF PATCHING

Patching combines requests with the patching stream. When a request misses the first part of a previous stream, it shares the rest of the stream and the server issues a patching stream to make up the request.

Patching is better than batching since it satisfies requests immediately. However, it is more complex and many version of patching have been invented [18, 14, 10, 15]. The most efficient patching method is the recursive patching where the patching streams are generated recursively to minimize the bandwidth requirement [20, 21]. A common assumption for patching is the receive-two model where a client can receive at most two streams at any time. Although different versions of patching require different number of streams, the bandwidth requirement can be roughly modeled as $(\ln(\lambda_i L) + 1)$ streams for video i [20, 21].

With this model, the number of streams required can be obtained as follows. When $\lambda_i L < 1$, no request can be combined. Although many algorithms do not attempt merging with an existing stream that is already at least half over, it is possible that some requests can be combined partially when $\lambda_i L > 1$. We divide the entire video repository into two sets according to $\lambda_i L < 1$ and $\lambda_i L \geq 1$. Thus, the turning point v can be obtained as:

$$\lambda_v L = \frac{C}{v} aL = 1,$$

and

$$v = CaL.$$

Also, because the upper limit of v is N , we have

$$v = \text{Min}(CaL, N).$$

The number of streams required can be expressed as follow:

$$\begin{aligned} m &= \sum_{i=1}^v (\ln(\lambda_i L) + 1) + (1 - r)aL \\ &= \sum_{i=1}^v (\ln(\frac{C}{i}aL) + 1) + (1 - r)aL, \end{aligned}$$

where, $r = \sum_{i=1}^v \frac{C}{i}$.

When $CaL < N$,

$$m = CaL + \sum_{i=1}^v \ln(\frac{C}{i}aL) + (1 - r)aL \geq CaL.$$

Thus, the reduction rate RR is less than $\frac{1}{C}$. Same as batching, patching is not scalable when $CaL \leq N$.

When $CaL \geq N$,

$$m = N + \sum_{i=1}^N \ln(\frac{C}{i}aL).$$

The number of streams required increases with $\ln a$ and the system is scalable.

When α is not equal to 1, the value of v and r should be changed to:

$$v = \text{Min}((CaL)^{\frac{1}{\alpha}}, N)$$

and

$$r = \sum_{i=1}^v \frac{C}{i^\alpha}.$$

5. SCHEDULED VIDEO DELIVERY

A video delivery paradigm, named *Scheduled Video Delivery (SVD)* has been proposed in [22]. This paradigm utilizes the property that not all contents are needed at the request time. In many situations, people can plan ahead to obtain some content before it is actually used. In the SVD paradigm, users submit requests with specification of start time. A pricing scheme ensures that the user-specified start time reflects user's real needs. The SVD system combines requests to form multicast groups and schedules these groups to meet the deadline. With this paradigm, requests can be combined

to reduce the server load and network traffic. Furthermore, the traffic can be smoothed by shifting the peak-time traffic to non-peak time.

SVD extends a user's option from click-wait-see patterns to plan-ahead alternatives, being able to submit requests with timing specification. SVD scheduling has a different objective from many existing scheduling schemes. It does not aim at minimizing the waiting time. Instead, it focuses on meeting deadlines and at the same time combining requests to form multicasting groups. The SVD paradigm has a similar behaviors of batching since the average of the deadline is equivalent to a certain batching time. What making SVD different from batching is that the equivalent batching time decreases with increase of a , more details will be shown later. However, with SVD, all requests are scheduled to meet the deadlines. Even immediate requests with zero plan-ahead time are served without waiting.

The SVD paradigm makes efficient use of the system resources by fully utilizing the time interval between when the user requests the video and when the user actually views it. Here, we briefly describe the SVD specification and derive a formula for SVD performance. The latest start time (deadline) of a request is uniform distributed between time 1 and time T_d . We need to obtain the equivalent batching time T before the formula for batching can be applied to compute the performance of SVD.

The requests arrive at different time with different deadlines. If no time deadline falls between time t_0 and time $t_0 + t_t$, no stream needs to be issued. The longest t_t is the equivalent batching time T which can be computed as follows. Assume a request arrives at time t_i with its deadline at $t_i + t_d$, where $1 \leq t_d \leq T_d$. Event B'_j is defined as the deadline of the request, arrived at t_i , falls into $t_i + j$. Since deadlines are evenly distributed, probability of event B'_1 is

$$P(B'_1) = \frac{1}{T_d}.$$

Consequently, define event $A'_i = \overline{B'_i}$ and the conditional probability $P(B'_x | A'_{x-1} \dots A'_1)$ is shown as below, standing for the deadline of a request, arrived at t_i , does not fall into $t_i + 1, \dots, t_i + x - 1$ and falls into $t_i + x$:

$$P(B'_x | A'_{x-1} \dots A'_1) = \frac{1}{T_d - x + 1}.$$

And,

$$\begin{aligned} P(A'_x|A'_{x-1}\dots A'_1) &= 1 - P(B'_x|A'_{x-1}\dots A'_1) = \\ &= 1 - \frac{1}{T_d - x + 1} = \frac{T_d - x}{T_d - x + 1}. \end{aligned}$$

Assume that for every time t_i , λ requests arrive with their deadlines at $t_i + t_d$, where $1 \leq t_d \leq T_d$. Event A_k is defined as none of deadlines of all λ requests, arrived at t_i , falls into $t_i + k$. The conditional probability of $P(A_x|A_{x-1}\dots A_1)$, that is, none of deadlines of all λ requests falls into $t_i + 1, \dots, t_i + x$ is

$$\begin{aligned} P(A_x|A_{x-1}\dots A_1) &= (1 - P(A'_x|A'_{x-1}\dots A'_1))^\lambda \\ &= \left(\frac{T_d - x}{T_d - x + 1}\right)^\lambda. \end{aligned}$$

Starting from time t , the arrival rate is λ . We define event C_k as that none of requests' deadlines is before time $t + k$. Therefore, for the requests arrived at t , probability of none of their deadlines is not earlier than time slot $t + k$ is

$$P(A_k A_{k-1} \dots A_1).$$

And, for the requests arrived at $t + i$, probability of none of their deadlines is earlier than time $t + k$ is

$$P(A_{k-i} A_{k-i-1} \dots A_1).$$

Thus, for all the requests arrived at $t, t+1, \dots, t+k-1$,

$$P(C_k) = P(A_1)P(A_2 A_1) \dots P(A_k \dots A_1)$$

Since $P(A_2 A_1) = P(A_2|A_1)P(A_1)$, we have

$$\begin{aligned} P(C_k) &= P(A_1)^k P(A_2|A_1)^{k-1} \dots P(A_k|A_{k-1}\dots A_1) \\ &= \frac{(T_d - 1)^{\lambda k}}{(T_d)^{\lambda k}} \frac{(T_d - 2)^{\lambda(k-1)}}{(T_d - 1)^{\lambda(k-1)}} \dots \frac{(T_d - k)^{\lambda}}{(T_d - k + 1)^{\lambda}} \\ &= \frac{1}{T_d^{\lambda k}} \prod_{i=1}^k (T_d - i)^\lambda \end{aligned}$$

Overall, define event D_k as, starting from t , none of requests' deadlines falls before $t + k$ and at least one of request's deadline falls into $t + k$. The probability of event D_k is,

$$P(D_k) = P(C_{k-1}) - P(C_k)$$

$$\begin{aligned} &= \left(\frac{1}{T_d^{\lambda(k-1)}} - \frac{(T_d - k)^\lambda}{T_d^{\lambda k}}\right) \prod_{i=1}^{k-1} (T_d - i)^\lambda \\ &= \frac{T_d^\lambda - (T_d - k)^\lambda}{T_d^{\lambda k}} \prod_{i=1}^{k-1} (T_d - i)^\lambda \end{aligned}$$

Average distance or the equivalent batching time T_i for video i is computed as,

$$T_i = \sum_{j=1}^{T_d} j \times P(D_j)$$

and the average distance for all videos or the equivalent batching T is

$$T = \sum_{i=1}^N \frac{C}{i} \times T_i.$$

The computation of value v is different from that for batching. Since the T value is different for each video, we have

$$v = CaT_v.$$

Consider the upper limit of v is N ,

$$v = \text{Min}(CaT_v, N).$$

After the value of v is obtained, we can use the equivalent batching time T and apply the formula for batching to compute the number of required streams for SVD, which is shown as follows:

$$m = \sum_{i=1}^v \frac{L}{T_i} + (1 - r)aL.$$

Finally, SVD can be combined with patching to further improve its scalability. Its formula for the number of required streams becomes:

$$m = \sum_{i=1}^v \left(\ln \frac{L}{T_i} + 1\right) + \sum_{i=v+1}^{CaL} (\ln(\lambda_i L + 1)) + (1 - r)aL.$$

6. DISCUSSION AND COMPARISON

When CaT is larger than N , the batching system becomes scalable. The value of batching time T is critical for the system performance. The longer the T ,

the early the system becomes scalable when the arrival rate a increases. Note that T can be larger than L and the number of streams required can be smaller than N . On the other hand, the longer the T , the longer the user must wait.

Patching provides immediate response while the turning point only depends on the video length L . Longer L provides more chances to combine. However, after CaL is larger than N , the number of required streams still slowly increases with a . Combined with batching, the number of required streams will eventually stay at some level.

SVD performs similar to batching, but provides a longer equivalent batching time T when a is small. Thus, the system becomes scalable early when a increases. Better than batching, the system can provide immediate response while it encourages users planing ahead. Its drawback is the same as patching, that is, after the turning point, the number of streams required still slowly increases. It can be improved by combining patching and SVD.

Next, we provide the analysis results generated from the formulas derived above. Most results show the number of streams when the arrival rate increases. Unless mentioned otherwise, the number of videos $N = 1,000$. The video length $L = 120$ minutes. The α value is set to 1.

Figure 1 shows the relation between the turning points and the number of videos for batching, patching, and SVD. The turning point is expressed by the a value where the system becomes scalable. Patching has a smaller a value than batching when batching time is 20 minutes. SVD becomes scalable much earlier than patching.

Figure 2 shows the performance of batching. Obviously, the longer the batching time, the better the performance. For vary short batching time such as one second ($T = 0.0166$), there is virtually no waiting time but scalability is poor since the arrival rate is hardly reach 450,000 per minute or 27 millions per hour. A longer batching time results in better performance. For example, when $T = 20$ minutes, the batching system becomes scalable when the arrival rate reaches 375 per minute or 22,500 per hour.

We now compare the performance of different methods in Figure 3. Figure 4 shows the details of the behavior of these methods in the non-scalable region.

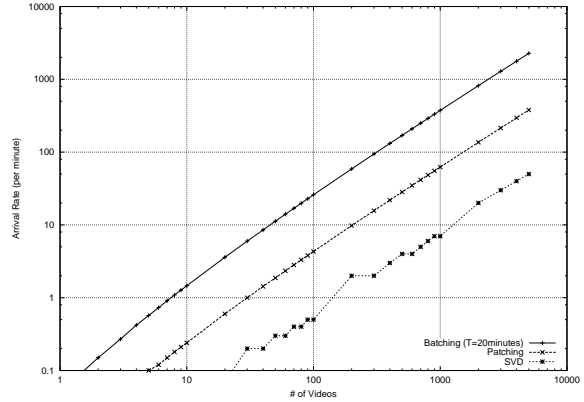


Figure 1: the relation between the turning point and the number of videos.

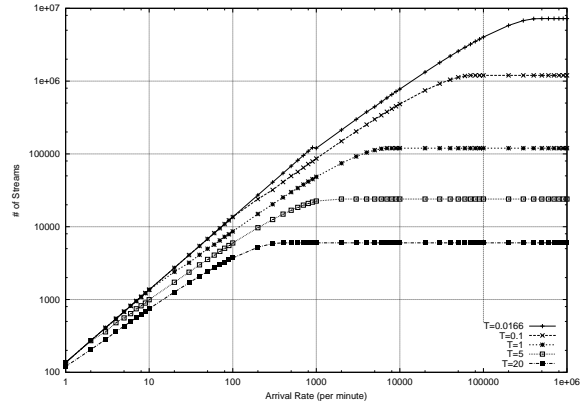


Figure 2: performance of batching.

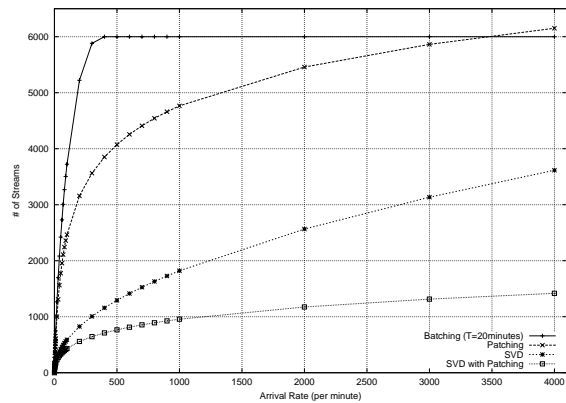


Figure 3: comparison of different methods.

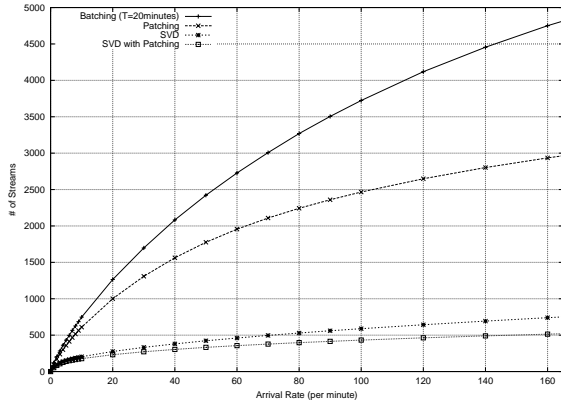


Figure 4: the behavior in the non-scalable region.

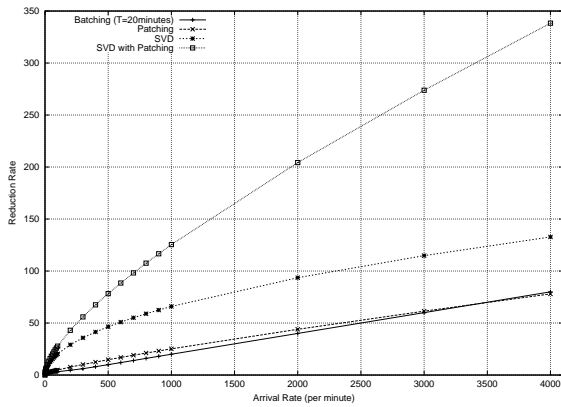


Figure 5: the reduction ratios of different methods.

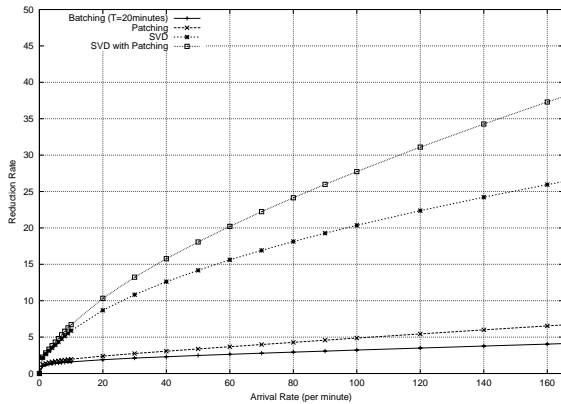


Figure 6: the reduction ratios in the non-scalable region.

Here, the batching time T is set to 20 minutes when comparing to other methods. Batching becomes scalable when $a = 375$. After that the number of streams remains 6,000. The turning point of patching is $a = 63$ and that of SVD is only $a = 7$. However, after the turning points the number of streams slowly increases. SVD with patching shows a much better performance. Its turning point is the same as SVD without patching but the number of streams increases much slower afterwards. The corresponding reduction ratio RR s of Figures 3 and 4 are shown in Figures 5 and 6, respectively. Patching becomes scalable when the arrival rate reaches 63 per minute or 3,780 per hour. When arrive rate reaches 1000 per minute or 60,000 per hour, RR is 25 and 4,766 streams are required. SVD with patching shows a good scalability. When arrive rate is 7 per minute or 420 per hour, only 157 streams are required. And when arrive rate reaches 1000 per minute or 60,000 per hour, RR is 125 and 956 streams are required. Thus, SVD with patching is a more scalable method.

The following figures show the sensitivity analysis. Figures 7, 8, 9, and 10 show the performance of $N = 200, 1,000, \text{ and } 5,000$ for batching, patching, SVD, and SVD with patching, respectively. A system with a large repository of videos is hardly scalable. A common video server delivers less than 1,000 video streams. Thus, it is difficult to make a video repository of more than 200 videos scalable. SVD with patching performs well. Less than 1,000 streams are needed for a video repository of 1,000 videos.

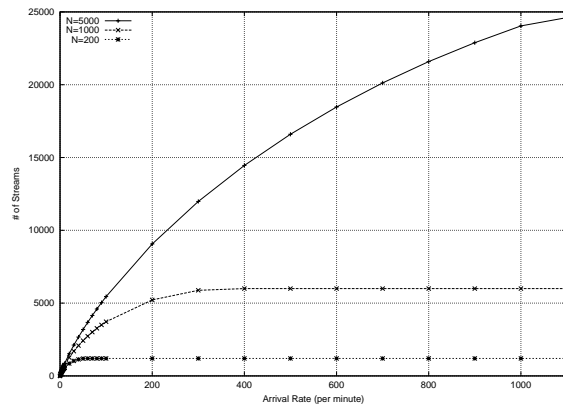


Figure 7: performance of batching ($T=20$ minutes) for different number of videos.

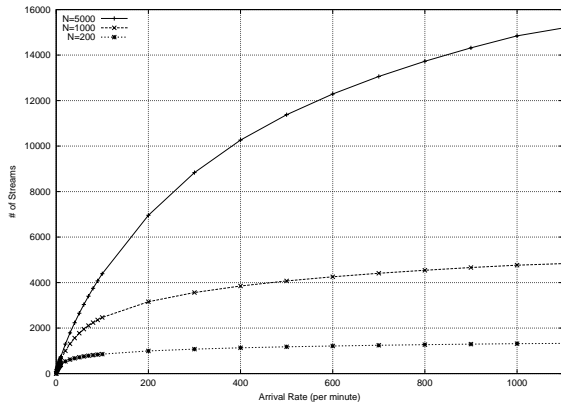


Figure 8: performance of patching for different number of videos.

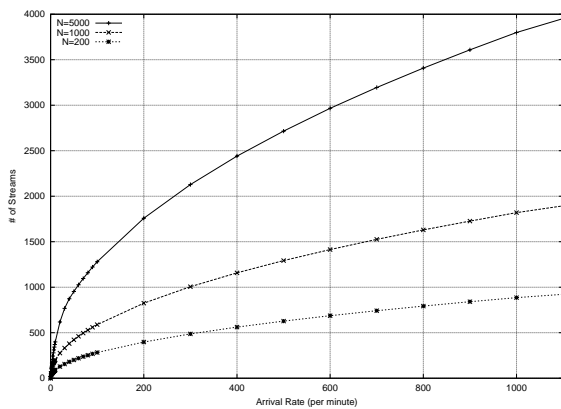


Figure 9: performance of SVD for different number of videos.

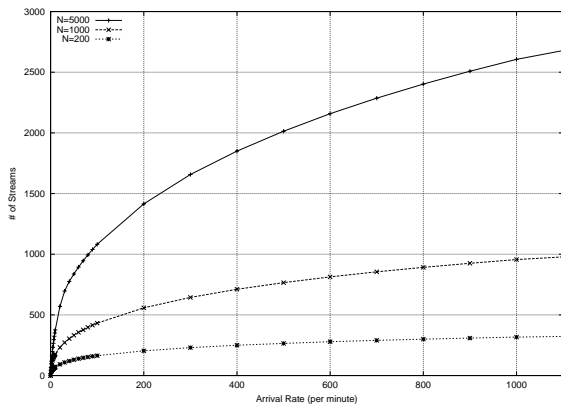


Figure 10: performance of SVD with patching for different number of videos.

Figures 11, 12, 13, and 14 show the performance of $L = 30, 60,$ and 120 minutes. The value of $L = 120$ minutes is the typical length of a movie. A smaller value of L makes the system more scalable.

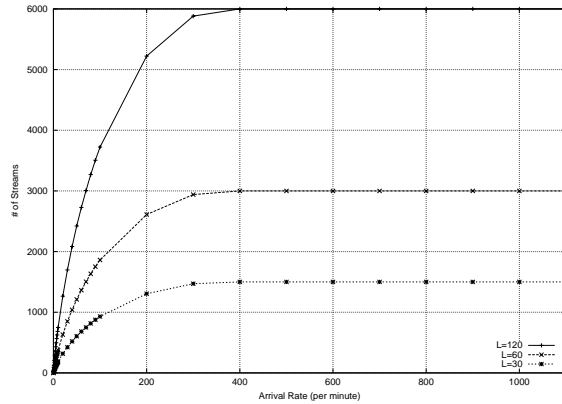


Figure 11: performance of batching ($T=20$ minutes) for different video lengths.

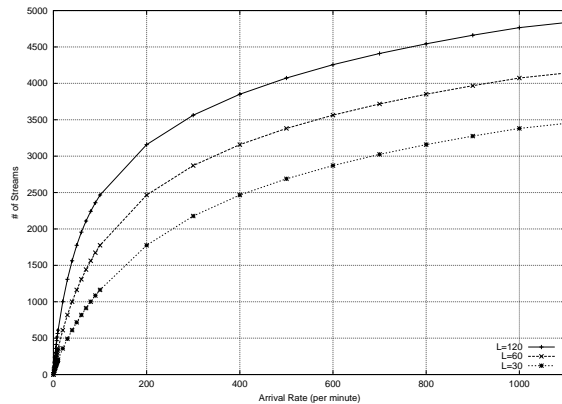


Figure 12: performance of patching for different video lengths.

Figures 15, 16, 17, and 18 show the performance of $\alpha = 0.6, 0.8,$ and 1.0 . In general, the larger the α value, the better the performance. For a smaller value of α , batching reaches its turning point earlier, requiring more streams when the arrival rate is low. For other methods, a smaller value of α results in more streams.

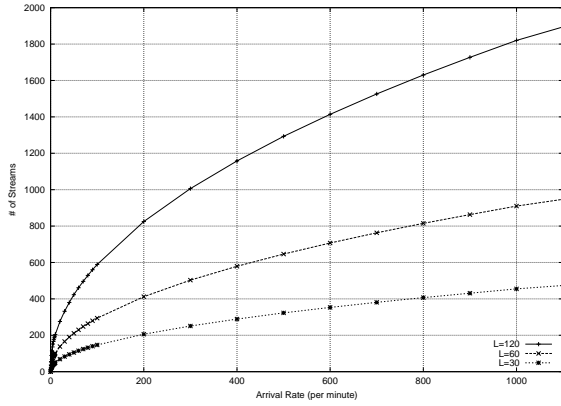


Figure 13: performance of SVD for different video lengths.

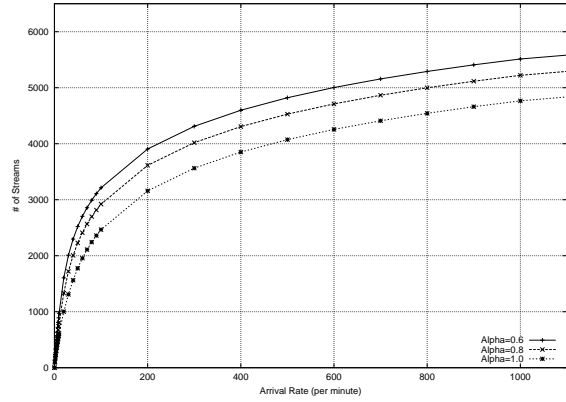


Figure 16: performance of patching for different α values.

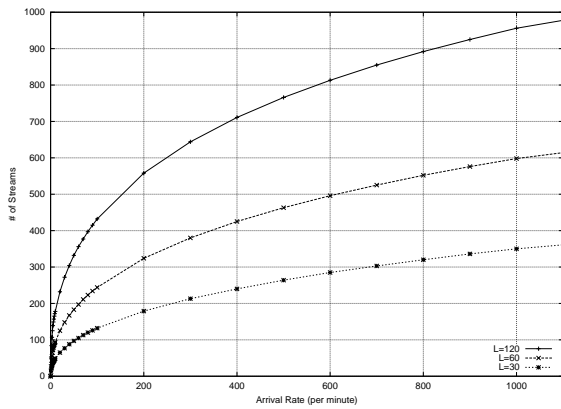


Figure 14: performance of SVD with patching for different video lengths.

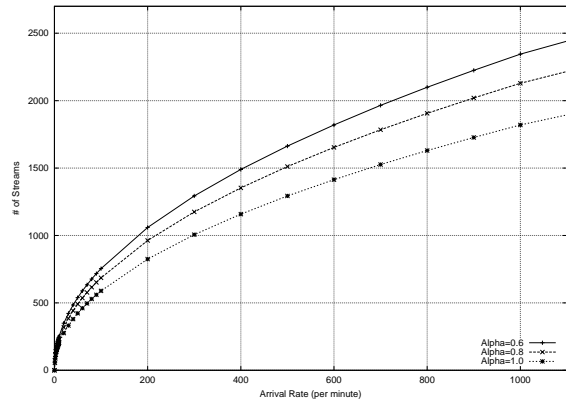


Figure 17: performance of SVD for different α values.

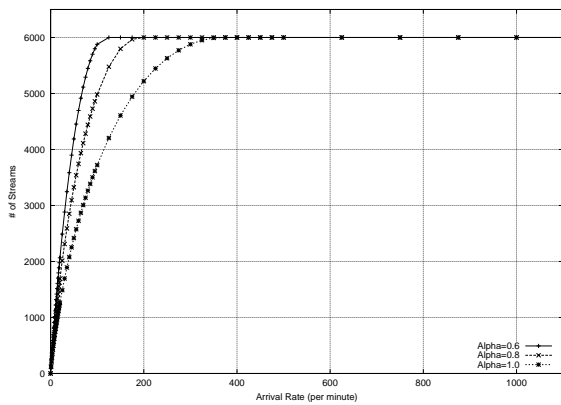


Figure 15: performance of batching ($T = 20$ minutes) for different α values.

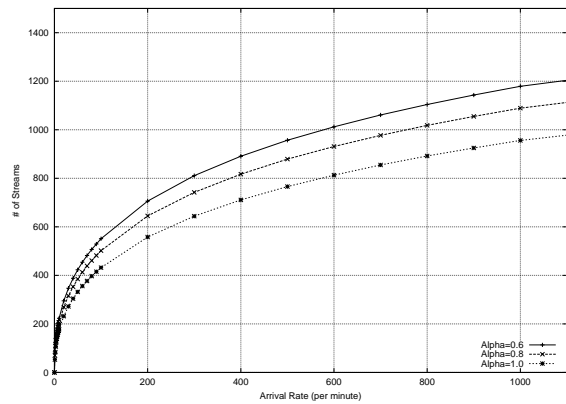


Figure 18: performance of SVD with patching for different α values.

7. CONCLUSION

Scalability of closed-loop on-demand video service has been studied in this paper. Performance models of batching, patching, SVD, and SVD with patching are presented. Analysis based on these models shows when a system can be scalable. The turning point depends on the size of video repository, the video length or batching time, and the arrival rate. The major conclusion is that the current methods are not scalable for a large repository of videos. They are scalable for only a small repository of videos. SVD with patching performs the best among these four methods. It scales to a quite large number of videos.

8. REFERENCES

- [1] H. C. D. Bey. Program transmission optimization, Mar. 1995.
- [2] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 4, no. 4, pp. 197–208, 1996.
- [3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *International Conference on Multimedia Computing and Systems*, pp. 118–126, 1996.
- [4] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *SIGCOMM*, pp. 89–100, 1997.
- [5] Y. Birk and R. Mondri, "Tailored transmissions for efficient near video-on-demand service," in *IEEE International Conference on Multimedia Computing and Systems*, 1999.
- [6] C. Aggarwal, J. Wolf, and P. Yu, "On optimal batching policies for video-on-demand storage servers," in *IEEE International Conference on Multimedia Computing and Systems*, June 1996.
- [7] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *ACM Multimedia*, pp. 15–23, 1994.
- [8] A. Dan, D. Sitaram, and P. Shahabuddin, "Dynamic batching policies for an on-demand video server," *ACM Multimedia Systems*, no. 4, pp. 112–121, 1996.
- [9] A. Dan, Y. Heights, and D. Sitaram, "Generalized interval caching policy for mixed interactive and long video workloads," in *SPIE's Conf. on Multimedia Computing and Networking*, pp. 344–351, Jan. 1996.
- [10] L. Gao and D. F. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *ICMCS, Vol. 2*, pp. 117–121, 1999.
- [11] L. Golubchik, J. C. S. Lui, and R. R. Muntz, "Reducing i/o demand in video-on-demand storage servers," in *Measurement and Modeling of Computer Systems*, pp. 25–36, 1995.
- [12] S. Sheu, K. A. Hua, and T. H. Hu, "Virtual batching: A new scheduling technique for video-on-demand servers," in *the 5th DASFAA*, Apr. 1997.
- [13] K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive video-on-demand service," *IEEE Journal of Selected Areas in Communications*, vol. 14, no. 6, pp. 1110–1122, 1996.
- [14] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *6th ACM Int'l. Multimedia Conf. (ACM Multimedia '98)*, pp. 191–200, Sept. 1998.
- [15] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming, Proc. 9th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'99), Basking Ridge, NJ, June 1999.
- [16] L. Gao, Z.-L. Zhang, and D. Towsley, "Catching and selective catching: Efficient latency reduction techniques for delivering continuous multimedia streams," in *7th ACM Int'l. Multimedia Conf. (ACM Multimedia '99)*, pp. 203–206, 1999.
- [17] S. Carter and D. Long. Improving video-on-demand server efficiency through stream tapping. In *ICCCN 97*, pages 200–7, Las Vegas, NV, USA. IEEE Computer Society Press, 1997.
- [18] D. L. Eager, M. K. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for video-on-demand servers," in *7th ACM Int'l. Multimedia Conf. (ACM Multimedia '99)*, pp. 199–202, Nov. 1999.
- [19] G. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley, 1949.
- [20] J. E.G. Coffman, P. Jelenkovic, and P. Momicilovic, "Provably efficient stream merging," in *Sixth International Workshop on Web Caching and Content Distribution (WCW'01)*, Mar. 2001.

- [21] A. Bar-Noy and R. Ladner, "Competitive on-line stream merging algorithms for media-on-demand," in SODA01, 2001.
- [22] M. Wu, S. Ma, and W. Shu, "Scheduled video delivery for scalable on-demand service," in International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), May 2002.