

Scheduled Video Delivery

— A Scalable On-Demand Video Delivery Paradigm

Min-You Wu, Su-Jun Ma, and Wei Shu

Department of Electrical and Computer Engineering
The University of New Mexico

Abstract — Continuous media, such as digital movies, video clips, and music, are becoming an increasingly common way to convey information, entertain and educate people. However, limited system and network resources have delayed the widespread usage of continuous media. Most existing on-demand video services, such as VoD, batching and patching, are not scalable. Although the Near VoD service is scalable, it can provide only dozens of videos. In this paper, we propose a scalable and inexpensive video delivery paradigm, named *Scheduled Video Delivery (SVD)*. In the SVD paradigm, users submit requests with specified start time. The SVD system combines requests to form multicasting groups and schedules these groups to meet the deadline. 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. SVD provides scalable on-demand video service with much less channels.

1 Introduction

In the last decade, we have witnessed a rapid growth of continuous media traffic over the networked world. The characteristics of continuous media are different from traditional text-based or image-based files. Typically, continuous media imposes high bandwidth and real-time requirements. Although audio and video can be used to convey information, entertain and educate people, the use of media applications is not widespread yet. This is largely due to limited system and network resources. The current media delivery systems are most best-effort. Servers and networks are designed for peak time and the resource is underutilized during the non-peak time. The server and network capacities become the major concern of continuous media services. Since media is delivered upon requests, rigid playback deadlines coupled with resource constraints make continuous media delivery a challenging task.

Most existing video services, such as Video-on-Demand (VoD), batching and patching, are not scalable. The number of channels required, where a channel represents the bandwidth requirement to deliver one video stream, is proportional to the number of clients when serving a large number of videos. In VoD, the number of channels is the same as the number of requests. The batching scheme is able to reduce demand on channels, but results in long waiting time. Furthermore, as will be shown in this paper, it is not scalable to a large number of video objects. The

patching scheme is scalable to popular videos as it requires $\log m$ channels for each video, where m is the number of requests for the video [1]. However, similar to batching, patching is not scalable to a large video repository. The only scalable scheme for video delivery is the Near VoD (NVoD), where the number of users is unlimited. However, each NVoD video needs 5 to 8 channels. Since video channels are limited resources, only dozens of videos can be serviced. It is desirable to find a scheme that is scalable to a large number of video objects. This objective can be made possible if considering the fact 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, which provides opportunities for the *Scheduled Video Delivery (SVD)* paradigm. With this paradigm, the traffic can be smoothed by shifting the peak-time traffic to non-peak time. Furthermore, requests can be combined to reduce server load and network traffic.

True VoD or patching provides the highest degree of interactivity. A video can be viewed immediately whenever the viewer wants to watch it. However, how many people need to view a video instantly? When people rent a video from a store, it takes minutes or hours to do so. Most people even do not have a chance to watch the video as soon as they get it. They rent a video for nighttime or weekend when they have time to relax. By taking advantage of this behavior, a video delivery system can be made scalable and inexpensive, and still flexible. Consider the cost of delivering a video to customers. Instant delivery to a single customer is costly. When a copy is delivered to many customers through broadcast or multicast, the cost can be reduced. Furthermore, minimizing the delay is a difficult task. A combination of disk scheduling, complex buffer management, and network congestion control is required. Also, performance cannot be guaranteed during the peak time. Interactive operations such as *fast-forward* and *jump* are difficult to provide with minimal delay. A slight relaxation of the delay requirement will simplify everything and reduce the delivery cost. Even a five-minute relaxation of real-time requirements can make video delivery scheduling much easier. A longer deadline would be even more beneficial. For example, when a customer makes the request two hours before the viewing time, the requests for the same object during this two hours can be combined and delivered together. Also, the video could be delivered during non-peak time too. When a video is stored in user's computer or STB, implementation of interactive operations becomes trivial. Furthermore, relaxing the real-time streaming requirement also eases fault-tolerance and heterogeneous network issues.

The SVD paradigm makes efficient use of the system resources by fully utilizing the time interval from when the request is made to when the user actually views the requested video. The large-scale use of continuous media applications demands developments in two directions: increasing resource provision and reducing resource consumption. With the SVD paradigm, many peak-time requests can be placed and executed ahead of time to utilize non-peak time server access and network bandwidth. It amounts to adding more underlying infrastructure to meet higher demands at peak-time; therefore, increasing provision of resource without incurring hardware costs. In the SVD paradigm, requests that are submitted ahead of time can provide a great opportunity for efficient scheduling and implicit combining, thereby reducing the overall resource consumption. In addition to multicasting or broadcasting, which reduces the resource requirement by combining requests across space, SVD combines requests across time.

This paper will describe the SVD architecture. Related researches are discussed in Section 2. The SVD architecture is proposed in Section 3. In Section 4, we discuss incentives and pricing for SVD. The system and scheduling are described in Sections 5 and 6, respectively. The analysis of scalability of the SVD scheme is given in Section 7. The experimental results are presented in Section 8. Finally, a discussion is provided in Section 9 and the paper is concluded in Section 10.

2 Related Research

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 video to individual users. It is not scalable because of its high bandwidth requirement. With the recent deployment of VoD systems in about a dozen of cities, we will soon know actual cost and performance of the VoD systems as well as their scalability.

VoD service is attractive since it is selective and responsive [2, 3]. With SVD, the selectiveness is retained, but the responsiveness has been changed into plan-ahead and guaranteed arrival. This alternative can be justified in several situations. Users may be very happy to exercise planing disciplines. Communities can be educated to change their usage-patterns to conserve resources in a global environment. Pricing can be differentiated to provide further incentive. While VoD service can still be effective for short clips, SVD will be suitable for high-quality, full-length movies and education/training courseware applications.

It is critical to investigate new methods for scalable video delivery. Two approaches for scalable video delivery have been developed. The open-loop approach [4, 5, 6, 7, 8] requires no return path so it can be used for one-way cable systems, whereas the closed-loop approach [9, 10, 11, 12, 13, 14] requires a two-way system. The closed-loop system only delivers the requested video to users but the open-loop system continually broadcasts a video even if no one watches it. Normally, only dozens of videos can be provided simultaneously in an open-loop system. Two major schemes have been proposed for the open-loop approach. The first one simply rebroadcasts the same video in a fixed time interval, for example, every 20 minutes. It results in a long average waiting time. The second scheme broadcasts the earlier segments of the video more frequently than the latter segments to reduce the waiting time [5, 6, 7]. It can only improve the waiting time if the viewer watches the video from the beginning, but one cannot jump to the middle of a video without initiating a separate video stream.

In the closed-loop approach, a number of methods, such as batching [9, 10, 11, 15, 16], patching [17, 13, 18], catching [19], stream tapping [20], and stream merging [21], are used to combine more requests to minimize the number of broadcast or multicast streams. Most of these methods assume some storage space being available in the client's machine. These methods require substantial bandwidth. Normally, five to ten video channels are required to provide one video object. Even worse, these channels broadcast the same video again and again, wasting significant

network resources. The same video channels could deliver a large amount of video objects if they were used in a different way. For instance, five video channels can deliver 60 two-hour videos during a 24-hour period. Many research works on the closed-loop approach emphasize merging more requests while minimizing waiting time, hence a common fact has not been noticed. That is, not all users want to view the requested video immediately. By utilizing this characteristics, video delivery scheme can become much more efficient.

Digital fountain [22] is a scalable approach to disseminating large files to many users. It is not suitable for video delivery with short deadlines. However, it can be used together with SVD to deliver videos. Other approaches to improving scalability are caching [23, 24, 25], mirroring [26, 27], or CDN [28]. These methods can be used with SVD to maximize the scalability.

Interactive Multimedia Jukebox (IMJ) reduces the channel requirement by scheduling requests to a certain time period, in the hope that other users will watch the already-scheduled videos [29]. Improved from IMJ, SVD presumes some user storage space, so the time requirement is further relaxed. In SVD, a video can be delivered before the user requested time, stored in the user's disks, and viewed any time after that. By moving the delivery time forward, it becomes simple to deal with the "dead period" problem [29] for the advanced reservation. The interactive operations also can be easily implemented when the video is already in the local storage.

Some other techniques have been used to reduce resource consumption. A policy using multiple service classes was also proposed in [30] which shares the same philosophy of the LPF algorithm described in this paper. Recent research proposes patching with cache [31, 32] to reduce the waiting time and improve performance. Patching can also be used in SVD scheduling to minimize the bandwidth requirement for requests with short deadlines.

Two commercial works, TIVO and Replay, can record the pre-scheduled broadcast streams. It facilitates the personal multimedia system. The SVD shares this idea with availability of storage and plan-ahead, but changes this passive scheme into an active one.

3 Scheduled Video Delivery

The SVD paradigm has a number of advantages compared to traditional VoD. First, SVD extends a user's option from click-wait-see patterns to plan-ahead alternatives, being able to submit requests with timing specification. Second, it enriches servers with capability of efficiently combining and scheduling requests. Third, it alleviates the peak-time overloading problem and improves utilization of limited bandwidth resources.

In the SVD paradigm, it is assumed that each user has sufficient storage for at least one video. Realistically, normal disks have sufficient space to store a few two-hour videos. A 40-GB disk, which is large enough to store a couple of MPEG-2 movie files, costs less than US\$50. We also assume a broadcast or a multicast scheme so that a

number of requests can be combined and delivered together. A cable (Hybrid Fiber/Coax) system can be used for this purpose. The Internet with multicast capacity will also satisfy this requirement.

In the SVD paradigm, a user makes a request of video with a *start time*. The server schedules a time slot for delivery of the video to the user's storage space on or before the start time. The start time can be as short as a few seconds and as long as 24 hours or more. A pricing scheme controls the overall cost for video delivery.

This paradigm also can be applied to content delivery from the original server to proxy servers. Some contents such as news and sports need to be delivered as soon as they become available. This kind of contents should have a short deadline. Other contents, such as movies, are sent to the proxy at a scheduled time with a long deadline. It gives the server scheduler a chance to smooth the load and traffic by taking advantage of different scheduling requirements.

The methods developed for videos can be applied well to non-video data, especially for large data files. This is because streaming a large file with a rate control reduces the network congestion compared to the best-effort approach.

4 Incentives and Pricing

The optimal efficacy can be achieved only if many requests are planned ahead. What will motivate users to do so? Why won't they always wait until the last minute to make requests? How to create incentives is crucial to the success of the SVD paradigm. Without an incentive mechanism, every user might request the shortest possible start time that does not reflect their real needs and the system resource could not be effectively utilized. Usage-disciplines and pricing policies could be combined to present users such an incentive system. These incentives must be carefully tuned so that user self-interests lead to an optimal overall resource utilization.

The usage-disciplines can be justified in several situations. Planning is a common discipline in our daily life, users could be very happy to practice planning if a proper infrastructure is available. Community can be educated to change their usage-patterns to conserve resource in the global environment. More important, if the peak-time on-demand requests most likely experience high rejection rate or low-quality video, users tend to plan-ahead for a guaranteed on-time, high-quality delivery.

The plan-ahead requests can also be encouraged by price differentiation. Various pricing schemes have been investigated by many researchers. The studies in [33] are based on a maximization process to determine the optimal resources allocation in a service-class sensitive pricing environment. In this work, the performance penalty received for requesting a less-than-optimal service class is offset by the reduced price of the service. That is, the lower price paid, the less quality service received. In the SVD paradigm, the plan-ahead requests still receive the high-quality

video in time. The works in [34] built a usage-sensitive pricing model. When users access resources they presumably take into account their own costs and benefits from usage, but ignore the congestion, delay, or exclusion costs that they impose costs on other users. This phenomenon is referred as *congestion externality*. It has been argued that congestion prices discourage usage when congestion is present, also generate revenue for capacity expansion. The SVD paradigm shares this similarity to penalize the immediate requests during congestion time, but also provides alternatives to place a request earlier and still to consume the media at the peak time. Wang and Schulzrinne [35, 36] have recently proposed a congestion-sensitive pricing dynamically depending on the availability of network resources. While this scheme triggers increasing of price when congestion is dynamically detected, SVD has placed an emphasis on shaping the bandwidth consumption in a larger time domain to provide a variety of price choices.

In the SVD paradigm, a pricing scheme is integrated to control the overall cost for delivering videos. When pricing reflects the real cost of video delivery, the resource utilization could be optimized. Both the provider and the consumer will benefit from this pricing model. Considering a broadcast/multicast based system as an example. The cost of the video delivery consists of what paid to the content producer, c_c , and the cost of video delivery, c_d . Normally, c_c is a constant at a certain time for a given video. However, c_d depends on the number of requests that can be combined for delivery. The number of requests to be combined is proportional to the plan-ahead time $t_{plan} = t_{start} - t_{arrival}$. Thus, the cost for delivery is inverse proportional to t_{plan} , $c_d = \frac{c_1}{\lceil \lambda \cdot t_{plan} \rceil}$, and the total cost is

$$C_t = c_c + \frac{c_1}{\lceil \lambda \cdot t_{plan} \rceil},$$

where c_1 is the cost to deliver a single video object. Since λ is the request arrival rate for the video, $\lambda \cdot t_{plan}$ is the number of requests to the video during the plan-ahead time. When this number is less than 1, no request can be combined, and the delivery cost is c_1 . A popular video will cost less to deliver even for a short plan-ahead time since its λ is larger than a non-popular video.

A sample price for a video is shown in Table I, assuming $c_c = \$0.5$, $c_1 = \$6$, and $\lambda = 0.1$ per minute. If you would like to watch this video at 7pm and make an immediate request, you will pay \$6.50. However, if you make a request before going home at 5pm, you will pay only \$1. Note that the prices for a 1-minute request and for a 10-minute request are the same since both have a little chance to combine other requests. Furthermore, the price of a video

t_{plan}	≤ 10 min.	20 min.	30 min.	1 hour	2 hours	5 hours	10 hours
price	\$6.50	\$3.50	\$2.50	\$1.50	\$1.00	\$0.70	\$0.60

Table I: Sample price for $\lambda = 0.1$.

depends not only on the plan-ahead time, but also on the popularity of the video. When the arrival rate increases, the price will be lower. Two more sample prices for $\lambda = 0.01$ and $\lambda = 1$ are shown in Tables II and III, respectively. A request on a popular video is charged a lower price because it will likely be combined to other requests and the

delivery cost is low. As an example, for a video of $\lambda = 1$, a request on the video with 6 minute plan-ahead time costs only \$1.50. On the other hand, people have to wait for a long time to watch a non-popular video with a low price.

t_{plan}	≤ 100 min.	200 min.	5 hour.	10 hour	20 hours	50 hours
price	\$6.50	\$3.50	\$2.50	\$1.50	\$1.00	\$0.70

Table II: Sample price for $\lambda = 0.01$.

t_{plan}	≤ 1 min.	2 min.	3 min.	6 min.	12 min.	30 min.	1 hour	2 hours
price	\$6.50	\$3.50	\$2.50	\$1.50	\$1.00	\$0.70	\$0.60	\$0.55

Table III: Sample price for $\lambda = 1$.

Providers have to confront issues of pricing and cost recovery. This price scheme will be published in advance or made known to the users during QoS negotiation. The longer the plan-ahead time, the more price incentives are presented. Users will select a plan-ahead time according to their real need. Some users may even be willing to wait longer for a better price, which further increases the system efficiency.

SVD provides flexibility to both providers and users. Providers are able to deploy the service incrementally. A few channels and low-cost servers can be deployed initially to test the demand of the video service. Since many requests can be combined, much more requests can be serviced compared to true VoD. With this limited deployment, some users who make short-time requests will be asked to change their requests to a longer plan-ahead time but given a lower price. More channels and servers will be deployed later if the provider identifies more users willing to pay higher price for short-time requests. Again, the high price discourages short-time requests or generates revenue for service expansion otherwise.

5 System Overview

There are three basic entities in the SVD paradigm: a Servicing Unit (SU), a Planing Unit (PU), and a Consuming Unit (CU). A SU consists of several modules: a scheduling module, a transmission module, and a content management module. A request for video delivery is initiated from the PU to the SU. It goes through QoS negotiation, and is admitted and scheduled at the SU. Based on the agreed delivery schedule, the content will be transmitted from the SU to the CU in time.

Content management module of SU. This module maintains the record of N video objects, O_1, \dots, O_N . For each object O_j we have:

- $l(O_j)$ playback length in minutes
- $r(O_j)$ average playback bit rate in Mbps
- $p(O_j)$ object popularity, from 1 to N

Though objects may have different bit rates and a single object may be variable-bit-rate (VBR), in this paper we assume all objects have the same constant bit rate. Objects with different bit rates or a variable bit rate will be considered in future work. The popularity of an object varies from time to time and is represented by a number, 1 is the highest and N is the lowest.

Planing unit. A request, once submitted by the PU, is assigned a unique request identifier i . For each request \mathcal{R}_i we have:

- $o(\mathcal{R}_i)$ object O_j requested, $o(\mathcal{R}_i) = j$
- $s(\mathcal{R}_i)$ the start time of consumption
- $a(\mathcal{R}_i)$ the request arrival time
- $q(\mathcal{R}_i)$ the plan-ahead time, $q(\mathcal{R}_i) = s(\mathcal{R}_i) - a(\mathcal{R}_i)$, which is equal to t_{plan}

The planing unit performs QoS negotiation with the scheduling module based on the available resources from SU.

Transmission module of SU. Considering the outgoing network bandwidth as well as the server's output capacity, the total available bandwidth W can be equally partitioned into M channels if all the streams have the same bit rate r , thus $M = W/r$. Once M , the number of channels, is fixed, the requests can be mapped onto the time domain of communication channels.

When the requests for the same object can be combined, an upper-bound of the bandwidth requirement for the object can be established based on the object length $l(O_j)$, bit rate $r(O_j)$, and plan-ahead time $q(\mathcal{R}_i)$. Assume all requests have the same $q(\mathcal{R}_i)$:

$$B = r(O_j) \times l(O_j) / q(\mathcal{R}_i) = r(O_j) / \beta,$$

where $\beta = q(\mathcal{R}_i) / l(O_j)$ represents a ratio of the plan-ahead time to the length of object. The higher the ratio, the less bandwidth is required. Note that this upper-bound of bandwidth requirement is independent of the total number of requests. This relationship indicates that longer plan-ahead time implies less bandwidth consumption.

Scheduling module of SU. The functionality of this module is to apply a scheduling algorithm to realize mapping

$$F_{alg}(R_k, C^k, P^k) \implies \begin{cases} C^{k+1} & \text{a new schedule} \\ P^{k+1} & \text{a new waiting pool} \end{cases}$$

where P^k is the current waiting pool and C^k the current schedule for each individual channel. Upon the arrival of request R_k , the scheduling module is invoked to change the schedule from C^k to C^{k+1} , as well as the waiting pool from P^k to P^{k+1} .

Requests for a same object could be combined into a group as long as they all meet deadlines. A group is the basic unit for scheduling. To maintain these groups, N pools, P_1, P_2, \dots, P_N , are formed. as $P_j = \{G_{j,0}, \dots, G_{j,n_j-1}\}$, where, n_j is the number of groups for object O_j ,

$$G_{j,v} = \langle U_{j,v}, e_{j,v}, c_{j,v}, t_{j,v}, d_{j,v} \rangle$$

for $0 \leq v < n_j$, and

- $U_{j,v} = \{R_i | o(R_i) \equiv j, a(R_i) \leq t_{j,v} \leq s(R_i)\}$ all requests in group $G_{j,v}$
- $e_{j,v} = \min_{R_i \in U_{j,v}} (s(R_i) + l(O_j))$ the deadline of $G_{j,v}$
- $c_{j,v}$ the channel that $G_{j,v}$ is scheduled to
- $t_{j,v}$ the time that $G_{j,v}$ is scheduled to
- $d_{j,v}$ the length of object O_j that $G_{j,v}$ has delivered

All requests for O_j that have not started their delivery should be in the same group. The deadline of $G_{j,v}$ is defined as the earliest deadline among all requests in the group, and $d_{j,v}$ maintains the length of object O_j that group $G_{j,v}$ has delivered.

Consuming Unit. This unit manages the user's resources and provides facility for a user to consume the media object requested. Examples of resources to be handled include storage space and network bandwidth. It may include the demodulation unit, decoding unit, and player. At the playback time, interactive functions such as fast-forward and jump can be performed as long as the media content resides in the local storage. In the case that the storage space is tight, buffer management should be taken care by the consuming unit too.

6 SVD Scheduling

The SVD paradigm requires a real-time scheduling system. The main objective of scheduling is to minimize the rejection rate. This in turn, can be described as two objectives:

- to schedule deliveries to meet the requested start time; and
- to combine as many as possible requests together to minimize the resource consumption.

The first objective is essential and the contribution of the other is twofold. First, combining requests reduces the resource requirement for a given load and minimizes provider's cost. Second, it leaves more space to help other requests meet their deadlines. When a request is submitted by the PU, the scheduling module of the SU will compute a schedule. If a feasible schedule exists, the request will be admitted; otherwise, it is rejected or further negotiated. The scheduling module guarantees to meet the requested start time, though the video object may be actually delivered earlier.

Since SVD scheduling is a real-time processing problem, the best-known Earliest Deadline First (EDF) algorithm can be applied with some modification. In addition to this real-time scheduling, EDF used in this case also combines as many as possible requests for the same object. This modified EDF algorithm with combining is named the MEDF algorithm as shown in Figure 1.

Upon arrival of request R_k for object O_j , we first check if there is any request group in P_j whose delivery has not started. If so, the newly arrived request can be combined into the existing one. In fact, if there is such a group, it is the only group for the object. As a result of combining, rescheduling is required if and only if $s(R_k)$ is earlier than the earliest start time of the existing requests. If there exists no such a group, a new group needs to be created and rescheduling is also needed in this case.

The scheduling procedure is applied to groups instead of requests. The number of groups is usually much less than the number of individual requests, especially for those popular objects. This multiple-channel scheduling uses a simple heuristic based on the earliest-deadline-first criteria. That is, the group with the earliest deadline will be scheduled first. It is a preemptive scheduling algorithm. The newly arrived request with earlier deadline can preempt a running process as long as the deadline of the process is met. When there exists no feasible schedule, the newly arrived request will be rejected. Note that t_{now} is the current time.

Although this algorithm can combine requests into groups, it does not minimize the number of groups. To combine as much as possible requests, the request groups for more popular objects should postpone their delivery as long as the deadline is met. Such a group has a better chance to combine more upcoming requests. An algorithm that delays delivery of popular objects can maximize combining. It is called the Least Popularity First (LPF) algorithm as shown in Figure 2.

The LPF algorithm has three major steps. First, the MEDF algorithm is applied to generate an initial schedule and at the same time, the scheduleability is tested. Second, the schedule is modified by pulling all delivery slots as late as possible without violating their deadlines. The popularity of an object is utilized at the last step. Based on the schedule produced by the second step, the request groups for less popular objects are moved forward if feasible.

For a newly arrived request R_k
 let $j = o(R_k)$ and $need_resch = false$
 if $\exists i, d_{j,i} \equiv 0$
 add R_k to $U_{j,i}$
 if $(s(R_k) + l(O_j)) < e_{j,i}$
 $e_{j,i} = s(R_k) + l(O_k)$
 $need_resch = true$
 else create a new group G_{j,n_j} in P_j for object O_j
 let $v = n_j$ and $n_j = n_j + 1$, add R_k to $U_{j,v}$
 $e_{j,v} = s(R_k) + l(O_j)$
 $need_resch = true$
 if $need_resch \equiv true$
 for every channel $c = 1, 2, \dots, M$, let $T_c^{(k+1)} = t_{now}$
 for every object $j = 1, 2, \dots, N$, and for every group
 $i = 0, \dots, n_j - 1$, in the ascending order of $e_{j,i}$
 find a channel c with the earliest available time $T_c^{(k+1)}$
 if $T_c^{(k+1)} + l(O_j) - d_{j,i} \leq e_{j,i}$
 $t_{j,i} = T_c^{(k+1)}$ and $c_{j,i} = c$
 $T_c^{(k+1)} = T_c^{(k+1)} + l(O_j) - d_{j,i}$
 else fail to schedule the new request R_k
 restore $C^{(k+1)} = C^k$ and $P^{(k+1)} = P^k$

Figure 1: F_{MEDF} : Modified Earliest Deadline First (MEDF) Algorithm

For a newly arrived request R_k , let $j = o(R_k)$,
 apply F_{MEDF} to obtain an initial schedule
 For each channel c , reconstruct an as-late-as-possible schedule
 let $A_c = \{(a_{c,i}, a'_{c,i})\}$ the available slots
 let $B_c = \{(b_{c,i}, b'_{c,i})\}$ the booked slots
 initially, $A_c = \{(T_c, \infty)\}$, $B_c = \{(t_{now}, T_c)\}$ and $L_c = \infty$
 mark its first scheduled group as *affinity* if $d_{j,0} > 0$
 For all *nonaffinity* groups with $c_{j,i} \equiv c$
 sort them in descending order of $t_{j,i}$
 shift group $G_{j,i}$ to its latest possible slot
 move slot $(t_{j,i}, t_{j,i} + l(O_j))$ from B_c into A_c
 let $t_{j,i} = \min(e_{j,i}, L_c) - l(O_j)$
 move slot $(t_{j,i}, t_{j,i} + l(O_j))$ from A_c into B_c
 let $L_c = t_{j,i}$
 For all *nonaffinity* groups in ascending order of $p(O_j)$,
 move the less popular stream forward
 move slot $(t_{j,i}, t_{j,i} + l(O_j))$ from B_c into A_c
 find a channel b with an earliest (a_x, a'_x) from A_b ,
 such that $a'_x \leq e_{j,i}$, and $a'_x - a_x \leq l(O_j)$
 move slot $(a_x, a_x + l(O_j))$ from A_b into B_b

Figure 2: F_{LPPF} : Least Popularity First (LPPF) Algorithm

The request groups for more popular objects tend to be left behind, increasing opportunities of combining.

An example is used to illustrate the both algorithms, F_{MEDF} and F_{LPF} . Here, let $M = 3$, $N = 5$ and all objects have the same length, $l(O_1) = l(O_2) = l(O_3) = l(O_4) = l(O_5) = 6$. However, they have different popularities, $p(O_1) = 1$, $p(O_2) = 2$, $p(O_3) = 3$, $p(O_4) = 4$, and $p(O_5) = 5$. There are ten requests listed in Figure 3.

With F_{MEDF} , R_1 is scheduled first. Requests R_2 , R_3 , and R_4 arrive at time 1. R_2 and R_3 with earlier deadlines are scheduled to processing. R_4 is admitted too. When R_5 arrives at time 3, it cannot be combined to $G_{1,0}$ since $G_{1,0}$ has started. As a result, request R_5 is rejected and the schedule at time 3 is shown in Figure 4. Then requests R_6 and R_7 arrive at time 4 and are scheduled as two new groups $G_{2,0}$ and $G_{1,1}$. At time 6, request R_8 is merged into $G_{1,1}$ to join with request R_7 . At time 7, request R_9 is merged into $G_{2,0}$ to join with request R_6 . However, there is no space left for R_{10} , which must be rejected. The schedule at time 7 is shown in Figure 5. Thus, with MEDF, two requests, R_5 and R_{10} , are rejected.

With F_{LPF} , R_1 is scheduled first when it arrives at time 0. Requests R_2 , R_3 , and R_4 arrive at time 1. R_4 and R_3 with less popularities are scheduled to process. R_2 is admitted, but its processing is postponed. When R_5 arrives at time 3, it is merged into $G_{1,0}$ which has not started yet. The schedule at time 3 is shown in Figure 6. As the time advances, request R_6 arrives and is scheduled as a new group $G_{2,0}$. Next, R_7 arrives and is merged into group $G_{1,0}$. At time 6, request R_8 arrives and is scheduled as a new group $G_{1,1}$, since group $G_{1,0}$ has started. At time 7, request R_9 is rejected and request R_{10} is combined into group $G_{1,1}$. The schedule at time 7 is shown in Figure 7. Thus, only one request, R_9 , has been rejected. As shown above, request R_9 has missed group $G_{2,0}$ for just a single time slot. If it is enforced to catch group $G_{2,0}$, the missed part can be patched. Adding patching to SVD scheduling algorithms will improve performance further.

7 Scalability of SVD

An analysis for scalability of the SVD paradigm is provided in this section. We assume ideal scheduling is used, that is, the system is not idle all the time and utilization is 100%. This will provide a performance bound for SVD. We will derive a formula for the number of channels at a given request arrival rate. This depends on request distribution, request arrival rate (a), the number of videos in the video repository (N), and the length of video. Without losing generality, we assume all video objects have the same length L .

Batching combines the requests arrived in some time period and satisfies them together. Since SVD shares behaviors of batching to some extent, we will derive the equivalent batching time T before the number of channels can be computed. We assume that the latest start time (deadline) of a request is uniform distributed between time 1 and time T_d .

R_i	R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9	R_{10}
$o(R_i)$	3	1	4	5	1	2	1	1	2	1
$a(R_i)$	0	1	1	1	3	4	4	6	7	7
$s(R_i)$	4	4	5	7	5	9	6	8	8	7
Deadline	10	10	11	13	11	15	12	14	14	13

Figure 3: The requests for the example to illustrate the algorithms.

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
channel 1	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	-	-
channel 2	-	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	-	-	-	-	-	-	-
channel 3	-	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	-	-	-	-	-	-	-

where, $G_{3,0} = \{ \langle R_1 \rangle, e = 10, c = 1, t = 0, d = 3 \}$, $G_{1,0} = \{ \langle R_2 \rangle, e = 10, c = 2, t = 1, d = 2 \}$
 $G_{4,0} = \{ \langle R_3 \rangle, e = 10, c = 3, t = 1, d = 2 \}$, $G_{5,0} = \{ \langle R_4 \rangle, e = 13, c = 1, t = 6, d = 0 \}$

Figure 4: The schedule at time 3 for MEDF

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
channel 1	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	-	-
channel 2	-	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	-
channel 3	-	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	-

where, $G_{5,0} = \{ \langle R_4 \rangle, e = 13, c = 2, t = 7, d = 0 \}$, $G_{2,0} = \{ \langle R_6, R_9 \rangle, e = 14, c = 3, t = 7, d = 0 \}$
 $G_{1,1} = \{ \langle R_7, R_8 \rangle, e = 12, c = 1, t = 6, d = 1 \}$

Figure 5: The schedule at time 7 for MEDF

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
channel 1	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	-	-	-	-	-	-	-	-
channel 2	-	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	-
channel 3	-	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	-	-	-	-	-	-	-

where, $G_{3,0} = \{ \langle R_1 \rangle, e = 10, c = 1, t = 0, d = 3 \}$, $G_{5,0} = \{ \langle R_4 \rangle, e = 13, c = 2, t = 1, d = 2 \}$
 $G_{1,0} = \{ \langle R_2, R_5 \rangle, e = 10, c = 2, t = 4, d = 0 \}$, $G_{4,0} = \{ \langle R_3 \rangle, e = 10, c = 3, t = 1, d = 2 \}$

Figure 6: The schedule at time 3 for LPF

time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
channel 1	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{3,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	$G_{2,0}$	-	-
channel 2	-	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{1,0}$	$G_{5,0}$	$G_{5,0}$	$G_{5,0}$	-	
channel 3	-	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{4,0}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	$G_{1,1}$	-

where, $G_{5,0} = \{ \langle R_4 \rangle, e = 13, c = 2, t = 1, d = 3 \}$, $G_{2,0} = \{ \langle R_6 \rangle, e = 15, c = 1, t = 6, d = 1 \}$
 $G_{1,0} = \{ \langle R_2, R_5, R_7 \rangle, e = 10, c = 2, t = 4, d = 3 \}$, $G_{1,1} = \{ \langle R_8, R_{10} \rangle, e = 13, c = 3, t = 7, d = 0 \}$

Figure 7: The schedule at time 7 for LPF

The probability that a request accesses video object O_j is

$$p_j = \frac{C}{j}, \quad \text{where } C = \frac{1}{\sum_{j=1}^N \frac{1}{j}}.$$

This is called the Zipf distribution [37]. Video object O_1 is the most popular and video object O_N is the least popular. For simplicity, we assume that the request arrival times are evenly distributed. When the request arrival rate for all video objects is a , the arrival rate for an individual video object O_j can be computed by

$$\lambda_j = ap_j = \frac{Ca}{j} \quad (1)$$

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 be derived.

Define event B'_i as the deadline of the request that arrives at t , falls into $t + i$, where $1 \leq i \leq T_d$. Since deadlines are evenly distributed, probability of event B'_i 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 the request, arrived at t , does not fall into $t + 1, \dots, t + x - 1$ and falls into $t + x$:

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

And, the conditional probability $P(A'_x | A'_{x-1} \dots A'_1)$ for an individual request arrived at t is,

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

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

$$P(A_x | A_{x-1} \dots A_1) = \left(\frac{T_d - x}{T_d - x + 1} \right)^\lambda. \quad (2)$$

Starting from time t , with arrival rate λ , 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 earlier than time slot $t + k$ is $P(A_k | A_{k-1} \dots A_1)$. Consequently, for the requests arrived at $t + i$, probability of none of their deadlines is earlier than time $t + k$ is $P(A_{k-i} | A_{k-i-1} \dots A_1)$. For all the requests arrived at $t, t + 1, \dots, t + k - 1$ and from Equation 2,

$$P(C_k) = P(A_1)P(A_2 | A_1) \dots P(A_k | A_{k-1} \dots A_1)$$

Since $P(A_2A_1) = P(A_2|A_1)P(A_1)$, we have

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

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

$$\begin{aligned} P(D_k) &= P(C_{k-1}) - P(C_k) = \left(\frac{1}{T_d^{\lambda(k-1)}} - \frac{(T_d - k)^{\lambda}}{T_d^{\lambda k}} \right) \prod_{i=1}^{k-1} (T_d - i)^{\lambda} \\ &= \frac{T_d^{\lambda} - (T_d - k)^{\lambda}}{T_d^{\lambda k}} \prod_{i=1}^{k-1} (T_d - i)^{\lambda} \end{aligned} \quad (3)$$

Average distance or the equivalent batching time T_j for video object O_j is computed as,

$$T_j = \sum_{k=1}^{T_d} k \times P(D_k) = \sum_{k=1}^{T_d} \left(k \times \frac{T_d^{\lambda_j} - (T_d - k)^{\lambda_j}}{T_d^{\lambda_j k}} \prod_{i=1}^{k-1} (T_d - i)^{\lambda_j} \right), \quad (4)$$

where λ_j is the arrival rate for video object O_j .

Given the equivalent batching time T_j , $\lambda_j T_j$ requests can be combined. When $\lambda_j T_j < 1$, no request can be combined. Since λ_j is monotonously decreased with j , the entire video repository can be divided into two sets by a *turning point* v . The first set, named as S_1 , is from video object O_1 to video object O_v where requests can be combined. The other set, named as S_2 , is from video object O_{v+1} to video object O_N where requests cannot be combined. The value of v can be obtained by

$$v = \text{Max}(j) | \lambda_j T_j > 1.$$

Without lossing generality, assume $\lambda_v T_j = 1$. Thus from Equation (1),

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

and

$$v = CaT_j.$$

Since $v \leq N$,

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

In order to understand importance of *turning point* v , define a parameter ρ to measure what percentage of requests belongs to the first set of videos, S_1 :

$$\rho = \frac{\sum_{j=1}^v \lambda_j}{\sum_{j=1}^N \lambda_j} = \frac{\sum_{j=1}^v \frac{1}{j}}{\sum_{j=1}^N \frac{1}{j}} = \sum_{j=1}^v \frac{C}{j}.$$

Let us find out requirements on the number of channels for the two sets of videos.

- S_1 . For every video object $O_j \in S_1$ during equivalent batching time T_j , we can combine $\lambda_j T_j$ requests since $\lambda_j T_j > 1$. Therefore, for O_j , at most L/T_j channels are required. The total number of channels required to serve video set S_1 is,

$$m_1 = \sum_{j=1}^v \frac{L}{T_j} \quad (5)$$

Notice that m_1 is independent of arrival rate.

- S_2 . For every video object $O_j \in S_2$, since $\lambda_j T_j \leq 1$, during equivalent batching time T_j , no request can be combined. Therefore, the total number of channels required to serve video set S_2 is equal to the number of requests which belong to S_2

$$m_2 = (1 - \rho)aL, \quad (6)$$

where aL is the total number of requests.

From Equations (5) and (6), the number of channels for request arrival rate a is

$$m = m_1 + m_2 = \sum_{j=1}^v \frac{L}{T_j} + (1 - \rho)aL \quad (7)$$

Equation (7) can be applied to both batching and SVD. Here, in SVD, T_j is the equivalent batching time for video object O_j , whereas in batching T_j is the specified batching time and could be the same for all video objects. Based on this equation the required number of channels for batching and SVD is plotted in Figure 8. Here, the parameters are set as $N = 1,000$, $L = 120$ minutes, $T = 20$ minutes, and $T_d = 1440$ minutes.

The average equivalent batching time for all video objects is as follows:

$$T = \sum_{j=1}^N \frac{C}{j} \times T_j.$$

Equation (7) can be simplified to

$$m = \sum_{j=1}^v \frac{L}{T_j} + (1 - \rho)aL = v \frac{L}{T} + (1 - \rho)aL. \quad (8)$$

The general scalability can be addressed based on value of ρ .

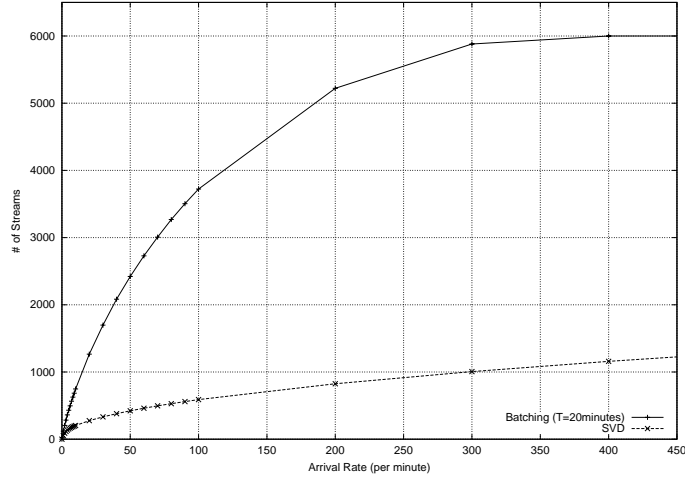


Figure 8: The required number of channels for batching and SVD.

- When $CaT < N$ and $\rho < 1$, from Equation (8)

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

The number of channels for the simple VoD is known as aL and C is a constant for a given value of N . Thus, the number of channels proportional to the number of requests. The system is not scalable under this condition.

- When $CaT \geq N$ and $\rho = 1$, from Equation (8)

$$m = m_1 = N \frac{L}{T}.$$

The number of channels is independent of the number of requests. The system is scalable to the number of requests.

For a given number of video objects, 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. The batching and SVD will result in the different division of the above two situations. Figure 9 shows the relation between the turning points and the number of videos for batching and SVD. For 5,000 video objects, batching needs 2,300 requests/minute to become scalable and SVD needs only 50 requests/minute.

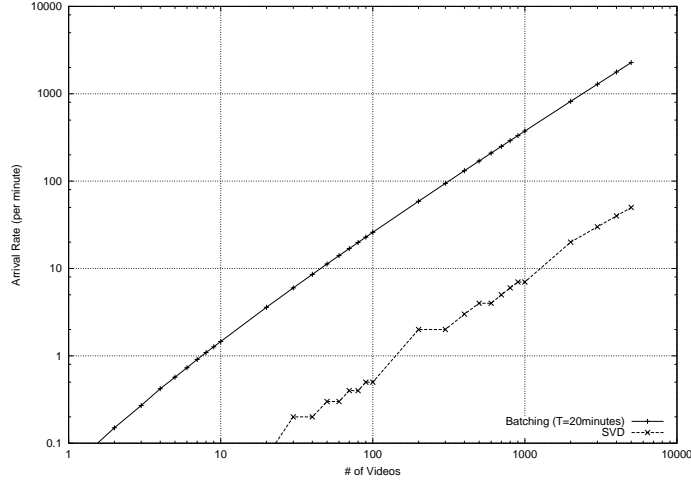


Figure 9: The relation between the turning point and the number of videos for batching and SVD.

8 Experimental Results

Our simulation environment is described here and preliminary results are presented. We show the results for various scheduling policies. The policies used in this simulation are listed below.

Scheduling Policy	Description
$F_{OnDemand}$	reject a request if it cannot be satisfied in one minute
$F_{Batching}$	force delay of delivery up to 10 minutes to wait for more upcoming requests and will reject the request if it cannot be satisfied in 12 minutes
$F_{Patching}$	the on-demand plus patching policy, rejecting a request if it cannot be scheduled
F_{MEDF}	the modified earliest-deadline-first policy, rejecting a request if it cannot be scheduled
F_{LPF}	the less-popularity-first policy, rejecting a request if it cannot be scheduled

The simulation parameters setting in this simulation are listed below. The number of video objects, N , is 1,000. For each request generated, a video is selected using a Zipf distribution [37]. The video length is chosen from a uniform distribution between 100 and 140 minutes with an average of 120 minutes. Plan-ahead time, $q(R_i)$, is uniform distributed between 1 minute and 1440 minutes (24 hours). The request arrival rate is set from 5 to 500 per minute.

Performance of various scheduling policies are compared against two metrics: (i) the required number of channels if no request is rejected; and (ii) the rejection rate for a fixed number of channels. Figure 10 shows the number of channels required for the five policies. The number of channels required by $F_{OnDemand}$ is proportional to the arrival rate. $F_{Batching}$ and $F_{Patching}$ require less channels. It can be seen that F_{LPF} for SVD performs much better than other

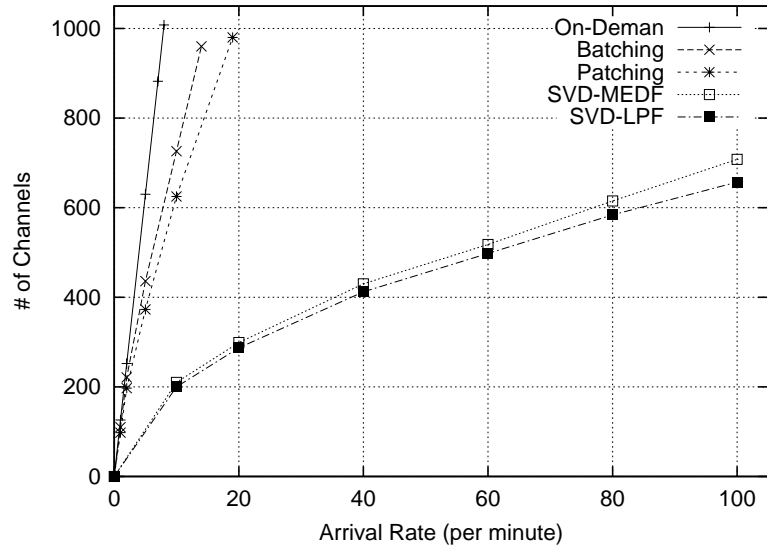


Figure 10: The Number of Channels Required.

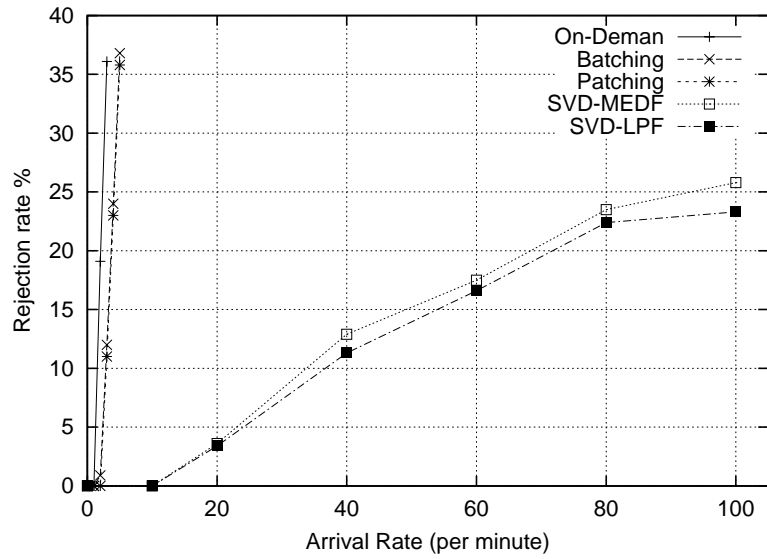


Figure 11: The Rejection Rates on 200 Channels.

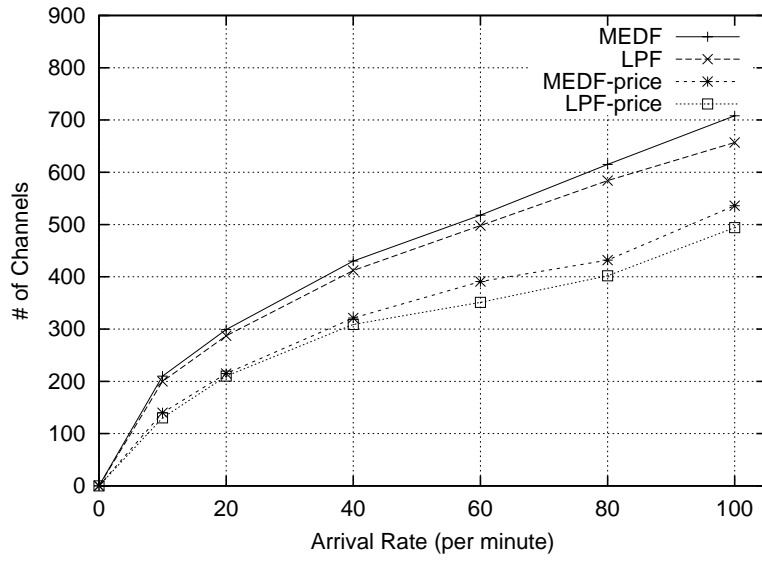


Figure 12: Comparison of number of channels influenced by pricing.

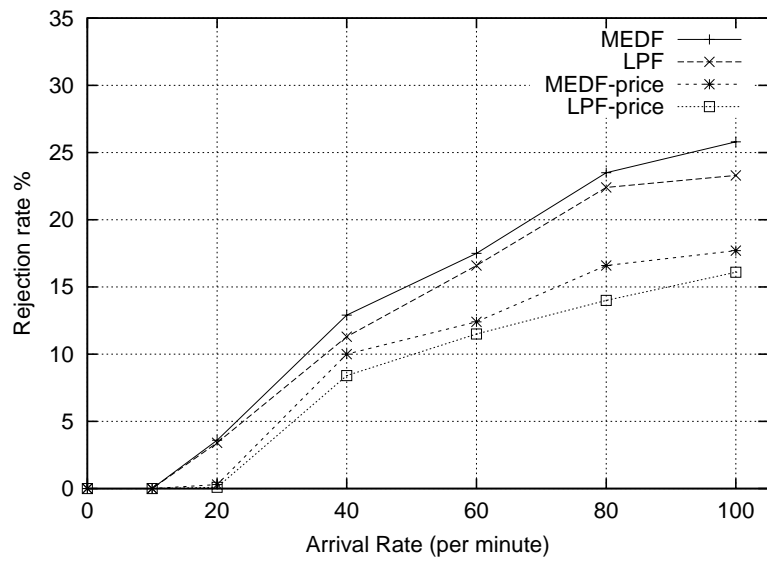


Figure 13: Comparison of rejection rates influenced by pricing on 200 channels.

policies. The number of channels required by the batching policy is 51% to 56% of that of the on-demand policy. The number of channels required by the patching policy is 46% to 52% of that of the on-demand policy. F_{MEDF} for SVD requires 708 channels when the arrival rate is 100/minute. F_{LPF} further improves the performance and requires only 657 channels. The comparison of theoretical and experimental results for SVD is shown in Table IV. The LPF algorithm generates schedules that are close to perfect scheduling.

Arrival rate (per minute)	20	40	60	80	100
Theoretical result	276	380	461	529	589
Experimental result (LPF)	287	412	498	584	657

Table IV: The channel requirement for SVD.

When the resource is limited, some requests will be rejected as the arrival rate increases. Figure 11 shows the rejection rates on 200 channels for these policies. $F_{Batching}$ and $F_{Patching}$ improve the rejection rates compared to $F_{OnDemand}$. For rejection rate of 5%, both of them can serve about twice of that for the On-Demand policy. SVD utilizes the user-provided deadlines and significantly reduces the rejection rate.

The above performance shows the situation when the user’s choice of plan-ahead time is not influenced by the price. In reality, when users are aware of price incentive, they tend to make the plan-ahead time as long as possible. In the following, we model this user behavior as that the request distribution is reverse proportional to the price, that is, $1/C_t$. Assuming $c_c = \$0.5$ and $c_1 = \$6$, the number of channels required with and without considering the influence of pricing for the F_{MEDF} and F_{LPF} policies are shown in Figure 12, where “MEDF-price” and “LPF-price” are for performance with pricing. Figure 13 shows their rejection rates on 200 channels. These figures show that the number of channels and the rejection rates can be further reduced when users are influenced by pricing.

Consider a HFC system, assume there are 200 channels as the system capacity which is equivalent to 25 analog cable channels for 3Mbps MPEG-2 videos. Assume six hours peak time per day [30] and the arrival rate in the peak time is five times higher than that in non-peak time [38]. With rejection rate of less than 5%, the $F_{OnDemand}$ service can serve up to 1,000 requests, the $F_{Batching}$ and the $F_{Patching}$ policies serve about 2,000 requests, and F_{LPF} about 35,000 requests in 24 hours. The SVD can serve about 35 times more requests compared to the on-demand service.

9 Discussion

The work presented in this paper aims at a framework of scheduled video delivery. The preliminary results show that SVD is a promising approach for truly scalable video delivery. SVD is suitable for the digital cable television network which is based on physical-broadcast scheme and has a fixed number of channels. SVD can also be applied to the Internet with multicasting capability. A number of issues to be investigated based on the SVD framework are

discussed here.

As we mentioned above, there are two essential requirements for deploying an SVD service, sufficient local storage space and the broadcast/multicast communication infrastructure. The storage requirement can be easily satisfied. The broadcast requirement is also straightforward in an HFC system. On the other hand, IP multicast has not been widely deployed yet and probably is not feasible in the near future. This may delay the application of SVD to the Internet. Fortunately, the server-based multicast on the overlay networks becomes a promising approach to be practical [39, 40, 41], which can be used as the multicast SVD service.

Another issue on the Internet is pricing. Internet service is traditionally free and people might not be willing to pay for the video service either. On the other hand, it is also a tradition that people pay for movie rental. As a substitution of movie rental, people would like to pay for the SVD service. The saving of on-line delivery should reduce the cost for the provider and the consumer should also pay less than that in a video rental store. In addition to convenience, the saving will turn consumers from the video rental store to the SVD service.

Incentive and pricing is an important component of SVD. The user's choice of plan-ahead time can be influenced by the price. In fact, not only the user's choice of plan-ahead time, but also the choice of videos can be influenced by the price, which will be studied in the future work.

Currently, we use the cost of delivery to establish a pricing model. This pricing model does not distinguish the peak time and non-peak time requests. A model needs to be established to take into account the relationship between the requesting time and the request's deadline. In fact, we do not know much about the user behaviors yet. Obviously some users will make advanced requests for better price or incentive, and others will make immediate requests regardless of price. A method is to be developed to gather user access patterns for future research.

10 Concluding Remarks

In this paper, we proposed a new video delivery paradigm. When all clients request immediate video delivery, SVD degenerates to the VoD service. By taking the user plan-ahead time into account, SVD can serve much more requests with the same resources. As a low-cost video delivery system, SVD has a number of advantages over the existing schemes. First, requests that are submitted ahead of time can provide a great opportunity for efficient scheduling and combining, as a result, reducing the overall resource consumption significantly. Second, peak-time requests can be executed ahead of time to utilize non-peak time server access and network bandwidth. It effectively increases the usable system resource. Compared to VoD, SVD provides not only immediate access of video, but also allows users plan-ahead to reduce the cost and price of video delivery. With SVD, on-demand video can be provided with much lower cost. Compared to the batching and patching approaches, SVD is able to combine more requests, reducing the

system resource requirement further. Compared to Near VoD, SVD does not send the same video repeatedly. The same number of channels can provide a large selection of video titles.

The SVD scheduling is different from any existing scheduling scheme. It does not aim at minimizing the waiting time like in most video scheduling algorithms, instead, it focuses on meeting deadlines and combining of requests to form multicasting groups. SVD defines a new class of scheduling algorithms. It is not a common real-time scheduling problem which does not combine requests.

Analysis and simulation results have been presented in this paper to show that SVD is a scalable and inexpensive approach to distributing and delivering videos. SVD requires much less resources to provide an on-demand video service compared to the true VoD. Integrating SVD with patching, and considering peak time behaviors will further improve scalability of video delivery.

References

- [1] J. E.G. Coffman, P. Jelenkovic, and P. Momcilovic, "Provably efficient stream merging," in *Sixth International Workshop on Web Caching and Content Distribution (WCW'01)*, Mar. 2001.
- [2] M. Wu and W. Shu, "Optimal scheduling for parallel CBR video servers," *Multimedia Tools and Applications*, May 2001.
- [3] M. Wu and W. Shu, "Efficient support for interactive browsing operations in clustered CBR video servers," *IEEE Trans. on Multimedia*, Mar. 2002.
- [4] H. C. D. Bey. Program transmission optimization, Mar. 1995.
- [5] S. Viswanathan and T. Imielinski, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Systems*, vol. 4, no. 4, pp. 197–208, 1996.
- [6] 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.
- [7] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *SIGCOMM*, pp. 89–100, 1997.
- [8] Y. Birk and R. Mondri, "Tailored transmissions for efficient near video-on-demand service," in *IEEE International Conference on Multimedia Computing and Systems*, 1999.
- [9] 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.
- [10] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *ACM Multimedia*, pp. 15–23, 1994.
- [11] 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.
- [12] 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.
- [13] L. Gao and D. F. Towsley, "Supplying instantaneous video-on-demand services using controlled multicast," in *ICMCS, Vol. 2*, pp. 117–121, 1999.

- [14] 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.
- [15] 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.
- [16] 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.
- [17] 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.
- [18] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming, Proc. 9 th Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSS-DAV'99), Basking Ridge, NJ, June 1999.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *SIGCOMM*, pp. 56–67, 1998.
- [23] S. 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.
- [24] D. L. Eager, M. C. Ferris, and M. K. Vernon, "Optimized regional caching for on-demand data delivery," in *Multimedia Computing and Networking*, Jan. 1999.
- [25] G. Dammicco and U. Mocci, "Program caching and multicasting techniques in vod networks," in *Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS)*, pp. 65–76, Sept. 1997.
- [26] J. Gafsi and E. Biersack, "Performance and reliability study for distributed video servers: mirroring or parity?," in *IEEE International Conference on Multimedia Computing and Systems*, pp. 15–23, June 1999.
- [27] S. D. Stoller and J. DeTreville, "Storage replication and layout in video-on-demand servers," in *Network and Operating System Support for Digital Audio and Video*, pp. 330–341, 1995.
- [28] Akamai, "Internet bottlenecks: the case for edge delivery services," tech. rep., Akamai White Paper, 1999.
- [29] K. Almeroth and M. Ammar, "The interactive multimedia jukebox (IMJ): A new paradigm for the on-demand delivery of audio/video," in *the Seventh International World Wide Web Conference*, Apr. 1998.
- [30] K. Almeroth, "Adaptive, workload-dependent scheduling for large-scale content delivery systems," *IEEE Trans. on Circuit and Systems for Video Tech.*, Feb. 2001.
- [31] J. Paris, D. D. E. Long, and P. E. Mantey, "Zero-delay broadcasting protocols for video on demand," in *7th ACM Int'l. Multimedia Conf. (ACM Multimedia '99)*, pp. 189–197, Nov. 1999.
- [32] M. A. Bing Wang, Subhabrata Sen and D. Towsley, "Proxy-based distribution of streaming video over unicast/multicast connections," Tech. Rep. 01-05, UMass CMPSCI Technical Report, 2001.
- [33] R. Cocchi, S. Shenker, D. Estrin, and L. Zhang, "Pricing in computer networks: motivation, formulation, and example," *IEEE/ACM Trans. Network*, pp. 614–627, 1993.
- [34] J. K. MacKie-Mason and H. R. Varian, "Pricing congestible network resources," *IEEE Journal on Selected Areas in Communications*, 1995.

- [35] X. Wang and H. Schulzrinne, "Performance study of congestion price based adaptive service," in *International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pp. 1–10, June 2000.
- [36] X. Wang and H. Schulzrinne, "An integrated resource negotiation, pricing, and QoS adaptation framework for multimedia applications," *IEEE Journal on Selected Areas in Communications*, 2000.
- [37] G. Zipf, *Human Behavior and the Principle of Least Effort*. Reading, MA: Addison-Wesley, 1949.
- [38] T. Little and D. Venkatesh, "Prospects for interactive video-on-demand," *IEEE Multimedia*, pp. 14–23, Fall 1994.
- [39] Y. Chawathe, *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, Department of EECS, UC Berkeley, Dec. 2000.
- [40] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr., "Overcast: Reliable multicasting with an overlay network," in *5th Symposium on Operating System Design and Implementation (OSDI)*, Dec. 2000.
- [41] P. Francis, "Yoid: Your own internet distribution," Tech. Rep. at www.aciri.org/yoid, UC Berkeley ACIRI Tech Report, Apr. 2000.