On-line Advancement of Transmission Plans in Video-on-Demand*

Juan Segarra
Universidad de Zaragoza
jsegarra@unizar.es

Vicent Cholvi
Universitat Jaume I
vcholvi@uji.es

Abstract

In this paper we detail an algorithm for the transmission of video-on-demand, which allows transmissions to be advanced dynamically when there is enough bandwidth. The effect is a transmission speed-up and consequently the freeing of bandwidth reservations in the future. This freed bandwidth can therefore be used by other transmissions. We evaluate two basic criteria for producing these advancements, and also detail how this scheme can be combined with other existing on-demand transmission methods. In our experiments we have combined it with an implementation of patching. *We compare the performance of our algorithm with this model, and also with a system without merging taken as a reference.*

We demonstrate that, during the high audience period, delays resulting from our approach are about 44% lower than without advancements, and we also state the importance of the criteria used for these advancements.

1. Introduction

Recent advances in data transmission technologies allow for the transmission of heavy data, which is the base of digital video-on-demand (VoD) delivery. Essentially, the two main aspects for study in video-on-demand systems are resources distribution [12, 14] and data transmission [3, 9, 5]. In this paper we focus on the latter. The data transmission problem is essentially one of how to guarantee a correct video transmission while, at the same time, offering the smallest possible serving times with the minimum bandwidth requirements.

Typical data transmissions are made through unicast protocols, which need a separate transmission for each client. This is obviously not suitable for video-on-demand since data streams need a large amount of bandwidth to be transmitted.

In order to save bandwidth, different broadcast/multicast-based mechanisms have been proposed. Perhaps the most widely known is the *pyramid broadcasting* scheme [3]. It broadcasts data in a "pyramidal" way, so that the beginnings of movies are broadcast very frequently, while later fragments are broadcast at a lower frequency (but nonetheless guaranteeing their reception before their deadline). Another method, the *patching* technique [5, 10], is based on the fragmentation of streams, so that multicast can be used more efficiently. The *piggybacking* solution [11] consists of varying the transmission rate of the streams in order to join streams of the same video that have been requested at different times. Finally, by using the *stream merging* technique [8], new requests make use of the streams already being transmitted. In this way, for a complete transmission, they only require a small part of the beginning of the stream to be retransmitted.

These transmission algorithms attempt to maximize the use of multicast capabilities, that is, they try to save bandwidth by transmitting streams to as many receivers as possible. This effort to save bandwidth implies that these transmissions are done over *lazy* transmission plans, i.e. streams are transmitted as late as possible. On the other hand, in these systems the load varies very quickly, depending on the time of day. This represents an important problem because streams are very long, and although some of them are requested before the request rate increment, their playing (and thus their transmission requirements) remains for some time during this period. Using simple batching, several studies have been conducted on this issue [1, 13], and the conclusion reached is that the fast request rate changes represent a performance decrement that is difficult to solve. However, when efficient techniques are used for transmission, this problem has not yet been analyzed.

In this paper we discuss a new transmission scheme that, unlike previous ones, allows the plan for transmitting a video to be changed on-line. When there is unused bandwidth, our system advances transmissions

* This study is partially supported by the CICYT under grant TEL99-0582 and by Bancaixa under grant P1-1B2000-12.

which had been planned for transmission in the future. This has a clear drawback, since it prevents from having lazy transmissions and thus from using multicast when transmission are advanced. However, it also introduces an important advantage. By using this approach, we try to "free" bandwidth reserved in the future so that it can be used by new requests. We discuss how to combine this scheme with other transmission methods and compare several policies concerning the selection of which transmission plans to advance. These experiments will show whether our approach effectively improves the system, or the drawbacks are too important.

The rest of our paper is organized as follows. In Section 2 we describe the scenario used. Our VoD delivery system is presented in Section 3, while in Section 4 we discuss how to combine our scheme with already existing ones. In Section 5 we present our results and, finally, our conclusions are discussed in Section 6.

2. Scenario

We use a fairly simple network architecture consisting of a central server, which stores all the videos. Users are connected to that server by means of a common link. The link used has a bandwidth of 1.5 Gbps downstream. A moderate bandwidth is also needed upstream in order to send user requests to the central server. We consider this to be a network devoted to VoD[1] and, therefore, our goal is to obtain the best results using the available resources.

The scenario we use has 15 000 users and contains 10 000 videos. We consider each video to have a total duration of 90 minutes and a constant bandwidth requirement of 2 Mbps. Videos are considered as being constant-bit-rate (CBR) for simplicity, despite the fact that, with minor modifications, our algorithm can work with VBR streams. However, with sufficient buffer capacity at clients, CBR transmissions are also acceptable.

In our approach to the video transmission problem, we studied performance over a 24 hour period. Previous studies on video demand rates [2, 7] have determined the behavior of these rates over this period, so our simulations work with a rate distribution provided by these studies. We also consider that a user makes a request only if he or she is not waiting for or receiving a video (i.e. nobody can receive more than one video transmission at a time). In Figure 1 we can see the request rate distribution obtained in a simulation having

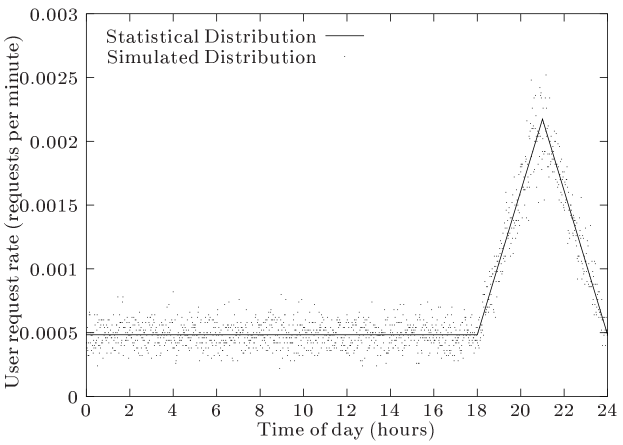

Figure 1. Request rate distribution over a 24 hour period.

one request per day for each user. This is what we assumed in our study (actual VoD trials show that the average number of user requests per week is about 2 to 3 [14]).

Using this request rate, videos are requested following a *popularity* factor, which represents the probability of a video being requested according to Zipf's law [4]. This distribution has been found to statistically fit video program popularities estimated through observations from video store statistics [6].

3. Advancement Algorithm

As mentioned above, in our approach the transmission plans can be modified. Thus, we can divide our transmission algorithm into two separate phases. The first one makes an initial reservation by developing transmission plans. The second one, which is our main contribution, tries to speed up the planned transmissions.

Our system works by using time-slots and videos divided into fragments. Each fragment has a transmission duration of one time-slot. This simplifies the bandwidth reservations and the management of transmissions. In this way, transmissions of video fragments can only start at the beginning of a time-slot, and finish at the end of the same time-slot. We use a time-slot of 60 seconds.[2] Additionally, the processing time of a request is also 60 seconds. This means that when a re-

1 These networks usually offer more services, but devoting a particular amount of bandwidth to each service is also usual.

2 Considering the request rate presented above, during the high audience peak there are no more than around 30 video requests per minute, so that the computational cost of such a time-slot is relatively low.

When a new video request is received:
Set the video start time to the next time-slot after the processing time.
While the full transmission is not reserved **do**
If there is a planned transmission of this video before or at its start time which has not yet started **then**
Join to the previously planned transmission using multicast.
Else if there is available bandwidth for the transmission to start at its start time **then**
Reserve it.
Else
Set the video start time to one time-slot further.
Return the video start time to the client.

Figure 2. Algorithm for the bandwidth reservation.

quest is received, it will take 60 seconds to process it. This process time is also considered in the speed-up phase of our scheme, explained below.

3.1. Reservation Phase

When a video is requested, the system must generate a transmission plan and provide the transmission start time of every fragment of the video. That is, it must reserve the required bandwidth during the required time-slots (maybe after some delay) guaranteeing that the full video will be played without interruption or loss of quality.

This phase can be performed using any existing transmission scheduler which fulfills this requirement. Perhaps the simplest transmission technique for this phase would be a system that considers videos as uninterrupted streams, and allocates their transmissions in the shortest time possible. Figure 2 shows this algorithm. Other usable techniques for this phase are discussed in Section 4.

3.2. Speed-Up Phase

The speed-up phase takes place every time-slot. When a new time-slot starts, the system tests whether all available bandwidth will be used in the next slot. If it finds that there will be unused bandwidth, it chooses video fragments which are to be transmitted later (i.e. anyone of the currently planned fragments) and transmits them during the next time-slot. That

When a new time-slot starts (i.e. each 60 seconds):
While there is unused bandwidth in the next time-slot **and** there are planned fragments available for selection **do**
Select anyone of the planned video fragments.
Move the selected fragment plan to the next time-slot.

Figure 3. Algorithm for the advancements.

is, the system reserves their transmission requirements for the next time-slot and cancels the current ones, so that these fragments will be transmitted during the next time-slot. In this way, we can accelerate the planned transmissions and consequently free the bandwidth used by these fragment plans in the future. Figure 3 shows the algorithm of the speed-up phase. Note that performance in this algorithm depends on the criterion for selecting which video fragments to advance. We present two such criteria in Section 5.

In order to have enough time to process advancements in each time-slot, we have also considered a process time of one time-slot in this phase. Note that the advanced plans will never use bandwidth needed by new requests, because new requests also have a processing time of 60 seconds, and thus will be planned at least two slots in the future.

4. Combination with Other Schemes

As stated above, the reservation phase can be performed using any existing transmission scheduler. The only requirement the technique chosen for use must fulfill is that, when a video is requested, it must generate a transmission plan and provide an instant response with the transmission start time of every fragment of the video. This is done by many of the transmission methods which guarantee the correct transmission of the stream, because reservations are fixed prior to transmission.

The algorithm shown in Figure 2 can be improved in many ways. One of the most highly developed transmission techniques is patching [5]. This technique uses videos divided into several stream units or fragments. Basically, when a new request arrives and the requested video is already being transmitted, the new client will receive, using multicast, the fragments of that transmission which have not yet been transmitted. Thus, the new client will only need a few fragments to obtain the complete video, and the request will be served by means of several streams of the same video, each one consisting of different fragments. Obviously, the client

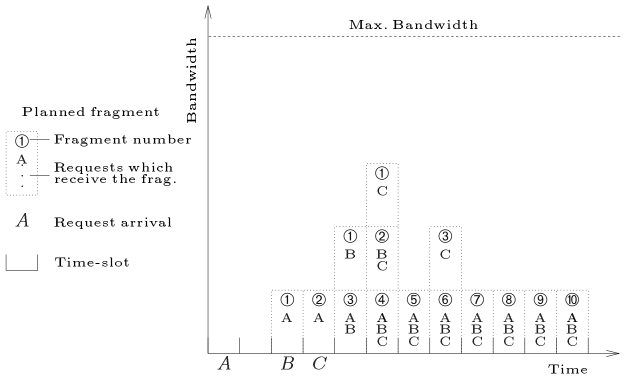

Figure 4. Example of our transmission plans. It considers one video only, consisting of 10 fragments. The request processing time is one time-slot. Request A is served by a new full stream. Request B receives some fragments served to request A (fragments 3 to 10) and the others (1 and 2) are retransmitted. Request C receives both fragments served to requests A (4 to 10) and B (2), and only fragments 1 and 3 are retransmitted.

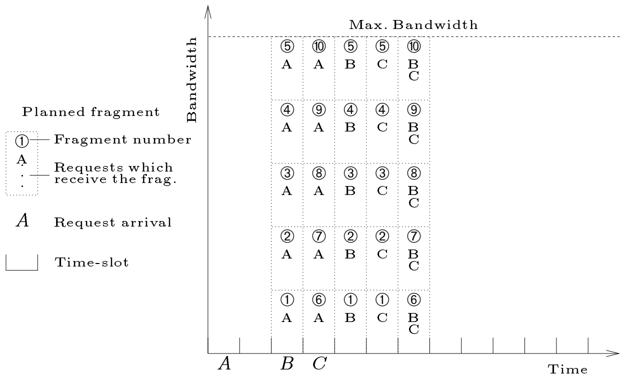

Figure 5. Actual transmission with advancements for the example in Figure 4. Request A is completely transmitted in two time-slots. Request B begins to be served in the same way, but request C arrives in the next time-slot. Thus, the remaining planned fragments for request B (6 to 10) will also be received by request C by means of multicast.

application has to be able to store and merge these partial streams to receive the complete video.

This transmission method fits our system very well because it already works with video fragments and merging.

Our patching implementation works by trying to place each transmission fragment first as a multicast transmission (without bandwidth costs) if possible, and with a bandwidth reservation at its transmission deadline (lazy transmission plan) otherwise. Both cases guarantee that each fragment will be transmitted without cuts or loss of quality. If this fails, it is because there is no previous transmission of this fragment to transmit in multicast mode, and also there is no available bandwidth in the tested time-slot. We therefore delay and test all its fragments again in one time-slot further until there is enough bandwidth for their transmission requirements (obviously, the video start time is also delayed). As a result, when a request is received, the system allocates a transmission plan for it as soon as possible, and returns an immediate response to the user with its start time. Given this instant response and the relatively short start-up delays obtained, we have not considered video rejections. However, considering them would simply involve canceling the reservations for the rejected transmissions.

Figure 4 shows an example of how our patching algorithm works. In Figure 5 we can see the actual transmission using advancements in combination with the patching example presented in Figure 4. It works as follows. Request A arrives and a (lazy) transmission plan is generated for it. When the second slot begins, the system detects that there will be unused bandwidth in the next slot, so it advances the transmission of fragments 2 to 5. The same behavior takes place in the next time-slot, and the system is able to transmit all remaining fragments. Request B is served in the same way because the previous request has been completely served. When request C arrives, fragments 1 to 5 of request B have already been served because of advancements, so they have to be transmitted again. At this moment, all fragments are again planned as lazy transmissions, fragments 1 to 5 to C and 6 to 10 both to B and C. Considering then that initial fragments are advanced first, fragments 1 to 5 are transmitted to C during the next time-slot, and finally fragments 6 to 10 are transmitted to requests B and C using multicast.

As can be seen, the actual transmissions (Figure 5) have used more bandwidth than the static transmission plans in Figure 4, but the transmission period has been reduced to 5 time-slots (50%).

5. Results

Our experiments have focused on two main aspects, the system delay in starting the transmission of the requested videos and the amount of bandwidth used.

The first results show the system performance with three different transmission algorithms. The *non-merging* represents a policy where there are no mergings in transmissions. That is, for example, if a video A has started its transmission in a slot t, and the same video is requested in slot $t + 1$, the new transmissions will not take advantage of the previous one, and the entire video will be transmitted again. We present this policy as a reference. The *patching* policy shows the system performance using our patching implementation without advancements, which we have outlined above. Finally, we present the advancement algorithm on patching using two basic criteria for the selection of fragments to advance: *more popular videos first* and *less popular videos first*.[3]

5.1. Delay Results

In Figure 6 we can see the starting delay (the time between a video request and the beginning of its transmission) throughout one entire day. It can be seen that during the low audience period all policies perform in the same way, by serving the received requests without any significant delay.[4] When the high audience period begins (see Figure 1), it can be seen how the *non-merging* policy is the first one to present significant delays, reaching up to 60 minutes. The *patching* policy is able to reduce the period where these delays appear, and also the amount of introduced delay. Finally, our algorithm presents very different behaviors depending on the selection criteria. The *more popular videos first* performs similarly to *patching*. On the other hand, the *less popular videos first* policy is capable of considerably reducing the delay period as well as the delay values.

This behavior can be explained as follows. The main feature of advancing fragments is that they will free bandwidth in the future. Thus, if we advance popular videos first, they will probably be requested again in a short time and they will have to be served again. In this case, the system will not gain any advantage from the advancements but it will make use of the bandwidth needed to perform them. On the other hand, if the system tries to advance the less popular videos first, future bandwidth will be freed and these videos are not likely to be requested in a short time. In this way, as

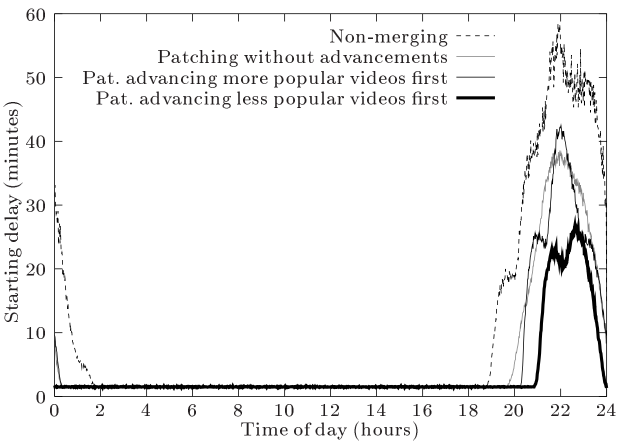

Figure 6. Starting delay over a 24 hour period.

this transmission will probably be unicast, these advancements will not prevent any multicast transmission.

Comparing our *less popular videos first* and *patching*, delays resulting from our approach are about 44% lower in the high audience period. The results obtained can be explained if we take into consideration that our approach reaches the high audience period with far more bandwidth available in the transmission link, and this allows a better absorption of the current load. This can be better seen in the next section.

5.2. Bandwidth Results

In this section we present our bandwidth usage results. Figure 7 shows that the main differences are located in the low load period, whereas all bandwidth is used when the load is high. The *non-merging* policy has a higher bandwidth utilization, and *patching* requires far less bandwidth. Our advancement policies perform at a middle point. The behavior of our approach during the low load period is due to its attempting to speed up transmissions which do not need it, since delays are already as low as possible. This prevents it from taking full advantage of the multicast capabilities and therefore uses more bandwidth.[5]

In our approach, the improvements in delays are obtained using more bandwidth during the very beginning of the high load period. Comparing it to *patching*, we can see how our algorithm reaches a bandwidth usage of 100% about one hour earlier, and it is during this time that the system really takes advantage of the

3 Both these policies select the initial fragments of the video first for advancements.

4 Working with time-slots, the mean minimum delay is half the time-slot, in this case 30 seconds, plus the 60 seconds processing time.

5 An obvious improvement would be to deactivate advancements during this period.

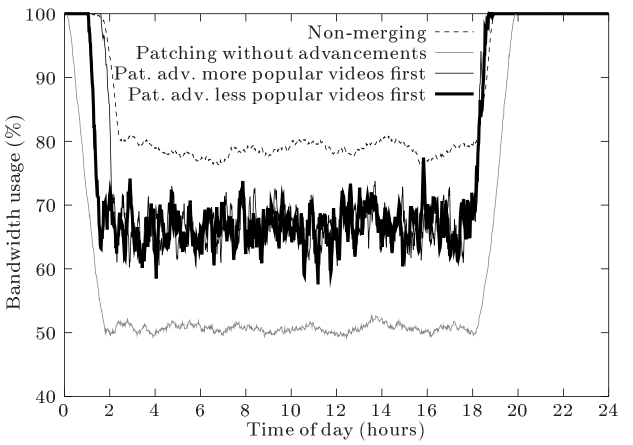

Figure 7. Bandwidth usage over a 24 hour period.

transmission speed-ups, thus freeing future bandwidth reservations.

Both advancing policies *more popular first* and *less popular first* have a similar bandwidth usage. The only difference is that, due to fragments advanced by *more popular first* have probably been requested again, it has more data to transmit and therefore its saturation period is longer.

6. Conclusions

In this paper, we have proposed a new transmission mechanism for VoD systems, and have tested two policies to use with it. This algorithm can be used with other reservation algorithms, such as patching. It basically adds a speed-up phase, which tries to advance transmissions that have been planned for transmission in the future. This partially solves the fast growth in request rate at the beginning of the high audience period, because the system reaches this point having more available bandwidth.

We have seen that it considerably reduces start-up delays as compared to the patching model without advancements, and we have also shown how the criteria used to select which fragments to advance affect very significantly the resulting performance.

In conclusion, we can say that advancements are a very effective way of reducing start-up delays, because they add the effects of high level prefetching to the transmission schemes used. Furthermore, reduction in start-up delays is mostly located over the high load period, when delays are longer. This is precisely the most critical period of time, because bandwidth is a bottleneck.

References

[1] K. C. Almeroth, A. Dan, D. Sitaram, and W. H. Tetzlaff. Long term resource allocation in video delivery systems. In *Proceedings of the IEEE Infocom*, pages 1333–1340, 1997.

[2] B. Atlantic. Fact sheet: Results of Bell Atlantic video services. Video-on-demand market trial. Trial results, 1996.

[3] Y. Birk and R. Mondri. Tailored transmissions for efficient near-video-on-demand service. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, volume 1, pages 226–231, Florence, Italy, June 1999.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of the IEEE Infocom*, Mar. 1999.

[5] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, pages 204–215, 1999.

[6] A. Dan, D. Sitaram, and P. Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4:112–121, 1996.

[7] P. de Haar et al. DIAMOND project: Video-on-demand system and trials. *European Transactions on Telecommunications*, 8(4):337–344, 1997.

[8] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for video-on-demand servers. In *Proceedings of the ACM Multimedia*, pages 199–203, 1999.

[9] D. L. Eager, M. K. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *Knowledge and Data Engineering*, 13(5):742–757, 2001.

[10] K. A. Hua, Y. Cai, and S. Sheu. Patching: a multicast technique for true vide-on-demand services. In *Proceedings of the ACM Multimedia*, pages 191–200, 1998.

[11] S.-W. Lau, J. C. S. Lui, and L. Golubchic. Merging video streams in a multimedia storage server: Complexity and heuristics. *ACM Multimedia Systems Journal*, 6(1):29–42, 1998.

[12] J. Segarra and V. Cholvi. Placement of storage capacity in distributed video servers. In *Proceedings of the IEEE International Conference on Communications*, New York, USA, Apr. 2002.

[13] J. Segarra and V. Cholvi. Simulations on batching in video-on-demand transmissions. In *Proceedings of the International Conference on Computational Science (Lecture Notes in Computer Science 2660)*, pages 453–462, Melbourne, Australia, June 2003.

[14] C. Vassilakis, M. Paterakis, and P. Triantafillou. Video placement and configuration of distributed video servers on cable TV networks. *Multimedia Systems*, 8:92–104, 2000.