Bandwidth Skimming: A Technique for Cost-Effective Video-on-Demand[*]

Derek Eager
Dept. of Computer Science
Univ. of Saskatchewan
Saskatoon, SK S7N 5A9 Canada
eager@cs.usask.ca

Mary Vernon
Computer Sciences Dept.
Univ. of Wisconsin
Madison, WI 53706
vernon@cs.wisc.edu

John Zahorjan
Dept. of Comp. Sci. & Eng.
Univ. of Washington
Seattle, WA 98195-2350
zahorjan@cs.washington.edu

ABSTRACT

Video-on-demand applications must consider the bandwidth limitations at the server, within the network, and at the client. Recent multicast delivery techniques have addressed the server and network bandwidth bottlenecks. These techniques, however, neglect consideration of client network access bandwidth limitations. Client bandwidth is often insufficient to permit streaming video of the quality that clients would desire, yet these new multicast delivery techniques require clients to be capable of receiving two or more transmission streams simultaneously, each at the playback bit rate. The reduction in the playback bit rate required to use these techniques implies substantially poorer display quality.

This paper proposes a new technique for on-demand delivery of streaming media that addresses this problem. The idea is to hold in reserve, or "skim", a portion of the client reception bandwidth that is sufficiently small that display quality is not impacted significantly, and yet that is nonetheless enough to support substantial reductions in server and network bandwidth through near-optimal hierarchical client stream merging. In this paper we show that this objective is feasible, and we develop practical techniques that achieve it. The results indicate that server and network bandwidth can be reduced to on the order of the logarithm of the number of clients who are viewing the object, using a small "skim" (e.g., 15%) of client reception bandwidth. These low server and network bandwidths are achieved for every media file, while providing immediate service to each client, and without having to pre-load initial portions of the video at each client.

Keywords: streaming media delivery, multicast, hierarchical stream merging, video-on-demand

1. INTRODUCTION

Application of video-on-demand is currently limited by economic considerations that are largely dominated by bandwidth issues, at both the client and server. At the client end, current network access bandwidths require extensive compromise in real-time playback quality: video frames are smaller than desirable, frame rates are lower, and compression levels are often high enough to produce visible artifacts. Even as client bandwidth is upgraded in the foreseeable future, new media files will emerge that require still higher data rates.

At the server end, the large number of potential clients, combined with the long duration of streaming media transmissions, translates to enormous server bandwidth requirements if each client is served with a dedicated stream. Considerable work has been done on reducing the server bandwidth required to provide on-demand access to popular continuous media files. However, there are significant drawbacks to each of the techniques proposed to date. The *batching* technique[6] reduces server bandwidth at the cost of undesirable latency to respond to client requests. The *piggybacking* technique[13,1,16] provides immediate service to each client, but merges clients during playback by speeding up and slowing down the client viewing rate. This technique requires the server to store (or compute in real time) extra encodings of the media data that is delivered most often. Furthermore, the maximum rate at which clients can be merged is bounded by the variation in viewing rate (typically ±5%) that can be tolerated by a client. Third, piggybacking cannot achieve *near-optimal* hierarchical stream merging because the early decisions to increase or decrease client viewing rate are frequently sub-optimal with respect to future unpredictable client request arrivals.[10] Other previous client merging techniques work only when client receive bandwidth is at least twice the bit rate needed for viewing.[21,2,5,14,15,7,11,3,8,17,20,9,4,10,12,18] These approaches cannot be applied when the media bit rate consumes more than half of the client bandwidth.

This paper proposes a new approach to making VOD systems practical. The key idea is to encode the media at a bit rate just slightly less than the (anticipated) client receive bandwidth. The receive bandwidth that is left unused during viewing (i.e., the "skim") is used to perform *near-optimal hierarchical stream merging*, a technique that has been shown to be highly effective in reducing server bandwidth when client receive bandwidth is two or more times the bit rate for viewing the file.[9,10] The key contribution of this paper is a set of techniques that use a small and highly adjustable skim to achieve very large

[*] This work was supported in part by the NSF (Grants CCR-9704503 and CCR-9975044) and NSERC (Grant OGP-0000264).

reductions in server bandwidth. For a given application, the skim can be "tuned" to provide the most desirable trade-off between playback quality and required server bandwidth.

The key problem to be solved to employ skimming is how to utilize the small bandwidth it makes available to perform stream merging. In this paper, we develop new delivery techniques that use limited excess client receive bandwidth to merge data transmission streams. The performance of the new bandwidth skimming techniques is evaluated using an optimal offline analysis (assuming all client request arrival times are known in advance and the technique can perform the best possible sequence of stream merges), and using the simple early merging heuristic[10] for near-optimal online merging (when future arrival times are not known). The principal results of evaluating the new techniques are:

1. Large reductions in required server bandwidth are achieved even for a small (e.g., 15%) skim.
2. Skimming increasing amounts of client bandwidth leads to diminishing incremental returns in server bandwidth reduction; for example, a 15% skim gives much of the benefit that is achievable when 50% of the bandwidth is skimmed.
3. To achieve much of the possible reduction in delivery cost requires client local storage capacity that is on the order of only 5-10% of the total data delivered during playback.
4. The new bandwidth skimming techniques provide more effective server bandwidth reduction than batching[6] over much of the system design space. Furthermore, for regions of the design space in which batching is fruitful, hybrid bandwidth skimming / batching techniques provide the bandwidth savings with a much smaller batching window size.

Bandwidth skimming is effective because with no skim and fixed client viewing rate, no stream merging can take place, resulting in server bandwidth requirements that are linear in the number of customers viewing the media. With even a small amount of skimmed bandwidth, however, stream merging is enabled and can reduce server bandwidth to on the order of the logarithm of the number of viewing customers. For unpredictable client request arrivals, the incremental benefits of skimming additional bandwidth for stream merging are greatest at exactly the transition from no possible stream merging to stream merging. Thus, skimming operates at the "sweet spot" of the stream merging techniques.

The remainder of the paper is organized as follows. Section 2 reviews the previous near-optimal hierarchical multicast stream merging technique that provides immediate service to clients at nearly the minimum achievable server bandwidth when client receive bandwidth is equal to twice the data playback rate. Section 3 describes the new client stream merging policies for bandwidth skimming, and Section 4 compares these policies using an offline analysis that elucidates the impact of the various design tradeoffs among them. Section 5 provides simulation results that evaluate how well the policies will perform in an online setting, over a wide range of the system design space. Section 6 concludes the paper.

2. NEAR-OPTIMAL HIERARCHICAL CLIENT STREAM MERGING

A number of recent papers have assumed that client receive bandwidth is at least two times the bit rate at which the desired video file has been encoded, and have proposed multicast delivery techniques that aggregate clients into successively larger groups that share the same transmission streams.[21,2,5,14,15,7,11,3,8,17,20,9,4,10,12,18] Of these techniques, the recent simple and near-optimal hierarchical multicast merging policies[9,10] provide the greatest server and network bandwidth reduction over a wide range of file request rates. In fact, the near-optimal hierarchical merging policies achieve nearly the minimum possible server and network bandwidth for providing immediate service to each client when client receive bandwidth is twice the data streaming rate. Furthermore, the hierarchical merging policies achieve this server and network bandwidth reduction without requiring initial portions of the video files to be pre-loaded at the client. One of these near-optimal hierarchical merging policies, called Earliest Reachable Merge Target (ERMT), is briefly reviewed in this section, in order to introduce the basic ideas of hierarchical multicast stream merging and to illustrate why required server bandwidth is proportional to the logarithm of the number of active client requests. This section also introduces the notation used in the figures that illustrate multicast merging techniques in the next section.

The essential elements of the recent multicast stream merging policies are: (1) each data transmission stream is multicast or broadcast so that any client can listen to the stream, (2) a client listens to more than one stream in order to accumulate data faster than the rate for viewing the object, thereby catching up to clients that started viewing the file earlier, and (3) once two transmission streams are merged, the clients listen to the same transmission stream(s) to receive the remainder of the data.

The key idea behind the ERMT technique is that transmission streams are merged as quickly as possible, assuming clients can receive two full transmission streams simultaneously and request arrival times are not known in advance. To accomplish this, each new client listens to its own new transmission stream, and also listens to the most recently initiated on-going transmission stream that it can merge with, if any. When two transmission streams merge, the clients continue listening on the stream that was initiated earlier, and the stream that was initiated later is terminated. The merged clients also start listening on the next most recently initiated stream that they can all merge with, if any.

The operation of the ERMT delivery technique is illustrated in Figure 1 for a given set of request arrivals for a given file. One unit of time on the x-axis corresponds to the time it takes to deliver the entire file at the bit rate required for viewing the object. One unit of data on the y-axis represents the total data for the file. The dashed lines represent data streams which, for the ERMT technique, always start at position 0 and always progress through the file at rate one unit of data per unit of time. The dashed line for a given stream may be (partially) hidden by a solid line. The solid lines show the amount of useful data that a client or group of clients has gathered as a function of time; the slope of a solid line is the rate at which the clients accumulate useful data. Solid lines that have unit slope are also streams (i.e., hidden dashed lines beginning at position 0.)

In the figure, requests are initiated by clients A, B, C, and D at times 0, 0.1, 0.3, and 0.4, respectively. Each new request initiates a new transmission stream. Client B also listens to the stream initiated by client A, accumulating data at rate two, and merging with client A at time 0.2. Client C listens to the stream initiated by client A and client D listens to the stream initiated by client C; however, client D merges with client C before client C can merge with client A. At the time clients C and D merge, both clients listen to the stream initiated by client C as well as the stream initiated by client A until both clients accumulate enough data to merge with the stream initiated by client A. This happens at time 0.8.

When a client is accumulating data at rate x greater than one, the local storage used to buffer data for future viewing increases at rate x-1 units of data per unit of time. Issues related to client buffer capacity are considered in Section 5.2.

Intuitively, due to the binary tree merging structure, hierarchical merging policies such as ERMT have required server bandwidth that is proportional to the logarithm of the number of active client requests. Previous work[9,10] shows that this is the case; for example, assuming Poisson request arrivals and an average of 1000 requests during the time that it takes to view the entire file, the average server bandwidth used for the ERMT delivery policy is on the order of 10 streams. Previous results[10] also show that ERMT achieves nearly the same server bandwidth savings as optimal off-line hierarchical stream merging (when client request arrivals are known in advance). These near-optimal savings are achieved because, at each point in time, each merged group of clients is attempting to catch the next earlier stream that it can merge with. The next section considers how to perform hierarchical merging when clients have excess reception bandwidth less than the file playback rate.

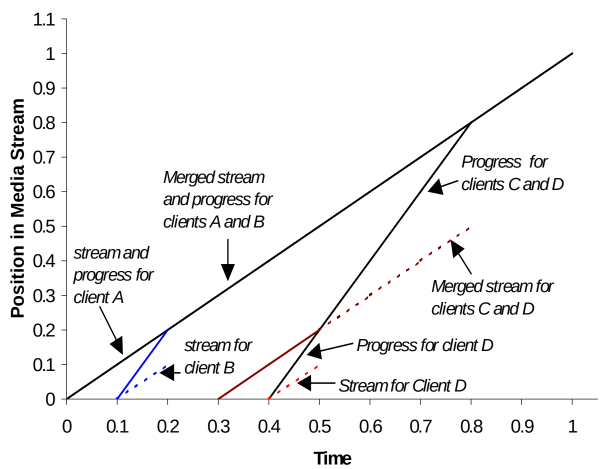

Figure 1: Example ERMT Operation

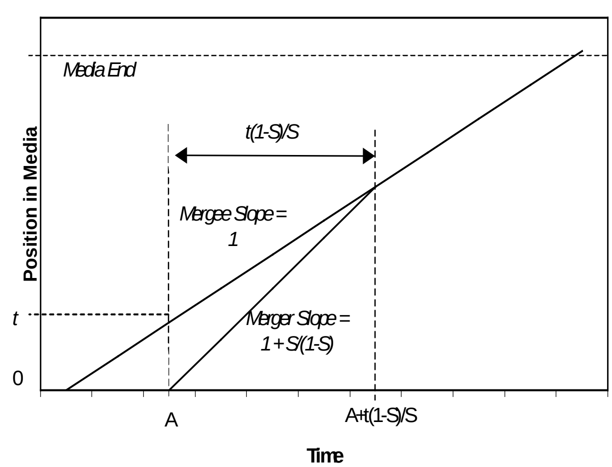

Figure 2: Basic Merging Concepts

3. BANDWIDTH SKIMMING POLICIES

This section defines several bandwidth skimming policies, each of which uses only a fraction of the media bit rate to effect hierarchical stream merging. The development of the policies considers the design trade-offs that must be made among (a) the total bandwidth required to effect a merge, (b) the number of simultaneous merges in which a single client can be engaged, (c) the complexity of the implementation, and (d) the number of distinct multicast channels required to support merging. Throughout the remainder of the paper, we use S to denote the fraction of client bandwidth that is skimmed.

3.1 Overview of Policies

The policies we consider share the following goals and characteristics, illustrated in Figure 2:

- If possible, a merge is initiated whenever a stream is "created" and there is an older stream in the system that "can be caught". By "created" we mean either the arrival of a new client request or the completion of the merge of two existing streams (resulting in a single, merged stream). By "can be caught", we mean that the merge can occur before the older stream is terminated.

- We refer to the stream being caught as the *mergee*, and the stream catching up as the *merger*. Note that in the online versions of the bandwidth skimming policies, it is desirable for a stream to simultaneously act as a merger (attempting to catch an earlier stream) and as a mergee (i.e., a target for a later stream). The playback point of a stream is the position in the stream that is being viewed by the client that arrived earliest among all clients who are receiving the stream.
- The mergee consumes (i.e., views) the media data at rate 1.0.
- A merger client uses its extra bandwidth to receive data at faster than the viewing rate; specifically, at rate $1+S/(1-S)$.
- As a consequence of the above two progress rates, the time required to merge two streams from a point at which their positions in the media file are separated by distance t is $t(1-S)/S$.

3.2 Partition

In the Partition policy, the merger listens to an increasing fraction of the mergee's delivery stream. To implement this approach, Partition requires that the skim percentage, S, be of the form $1/(m+1)$ for integer $m \geq 1$. Each rate 1.0 media stream is transmitted on m separate multicast addresses (or channels), each at rate $1/m$.

Figure 3 shows the operation of Partition when the skim S is 25%. (This is perhaps a larger than desirable skim percentage, but is convenient for use in the figure.) In this case, the extra client receive bandwidth is equal to 33% of the bit rate for viewing the file. Thus, the normal rate 1.0 stream is provided through three multicast channels of rate 0.33 each, and the skimmed bandwidth can be used to receive an extra rate 0.33 channel.

In the simple version of the policy, the server transmits separate rate 1.0 data transmission streams (each on three multicast channels) for the mergee and for the merger, throughout the merge. Each merger client goes through three distinct periods. During the first, it receives the *three* merger streams, at total rate 1.0, and uses its skimmed bandwidth to listen to *one* of the mergee's three streams. During the second period, it has already received the data that will be delivered in one of the merger streams during that period, and so needs to listen on only *two* channels to obtain the rest of the merger stream data. The merger's remaining bandwidth is used to snoop on *two* mergee channels. During the third period, only one channel is needed for data that is delivered for the merger stream, leaving full rate 1.0 bandwidth (three channels) for listening to the mergee stream. At the end of the third period, the merger has caught up to the mergee, and the merge is complete.

In the general case, for $S = 1/(m+1)$, m distinct periods are required, numbered 1 to m. Each period is of duration equal to the distance in playback position separating the merger and mergee at the beginning of the merge. During each period i, the merger devotes bandwidth i/m to snooping the mergee, and uses the rest of its bandwidth to obtain data from the merger stream. Note that a transmission stream can simultaneously be a merger and a mergee in the simple Partition policy.

A possible optimization of the Partition policy is to have the server only transmit the partial merger streams that the merger clients are actually listening to in each period. If this optimization is implemented, then the system must commit to the merge operation by the end of the first period, when less information is known about future client request arrivals than in later periods. However, the length of the first period in the Partition policy is equal to the amount of time for the two streams to commit the merge in the near-optimal hierarchical stream merging policies (e.g., ERMT) with skim equal to 50%. (See Figure 1.) Furthermore, the decision to commit the merge at the end of the first period is only a poor decision (relative to the simple Partition policy) if there is a newer stream than the merger stream that has not yet reached the end of its first period, but it would merge with the merger before the merger merges with the mergee in simple Partition. Thus, it seems likely that the server bandwidth savings per committed merge will outweigh the bandwidth loss due to poorer merge decisions, and we evaluate the performance of this optimized Partition policy in the experiments in Section 5 of this paper.

The principal drawback of the Partition policy is the number of multicast addresses or channels that must be used to support merging operations. Another disadvantage is the somewhat complex delivery schedule, which complicates policy implementation. Finally, the Partition policy only supports specific discrete values of the skim, which limits flexibility in trading off the size of the skim against the quality of the media stream. These drawbacks are largely eliminated in the Latest Patch policy discussed next.

3.3 Latest Patch

To reduce the number of multicast addresses and simplify the delivery schedule, each merger in the Latest Patch policy receives either one or two multicast streams for the duration of the merge operation, depending on the policy implementation. Figure 4 shows the basic merge operation of the Latest Patch policy, for the particular example where 25% of the client bandwidth is skimmed. The two possible implementations of the merge are discussed below.

In one implementation, the merger listens to its rate 1.0 stream as well as a new rate $S/(1-S)$ patching stream during the merge operation. These streams are shown as dashed lines in the figure. The Y-axis values of the dashed lines correspond to the intervals of the media file that are received from each source. The patching stream contains the latest data that is needed to reach the merge point (i.e., the interval [0.3,0.4] in Figure 4), so that all data is received by the time it must be displayed. The merger buffers the data from the patch stream, and begins viewing this data when the merge operation is complete.

The implementation that uses a separate patch stream provides the basis for the Shared Latest Patch and Discontinuous Patch policies that are described Sections 3.3 and 3.4. However, since each patch stream is received only by the clients of a single merger stream, there is a simpler implementation that yields equivalent performance. In this simpler approach, there is no separate patch stream – only a single merger stream that transmits data at rate 1+*S*/(1-*S*), but makes progress in the playback position at rate 1.0. As before, a merge is complete when the data received by the merger clients is equal to the rate 1.0 playback point of the mergee. Note that in either implementation, clients do not share reception of any stream until they have been merged. Furthermore, each stream can operate as both a mergee and a merger throughout the merge operation, allowing earliest possible merge points to be detected as late as possible relative to unpredictable client request arrivals.

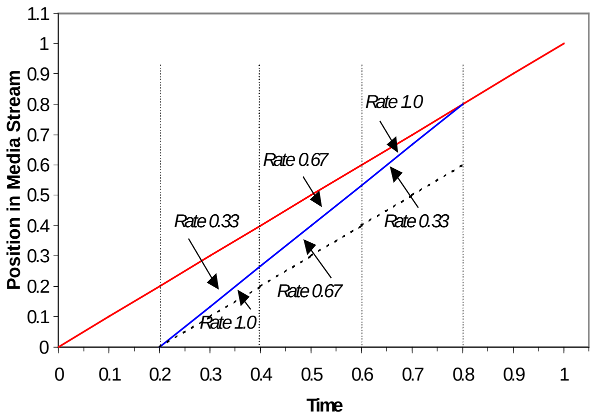

Figure 3: Partition
(The merger snoops as much of the data being sent to the mergee as possible.)

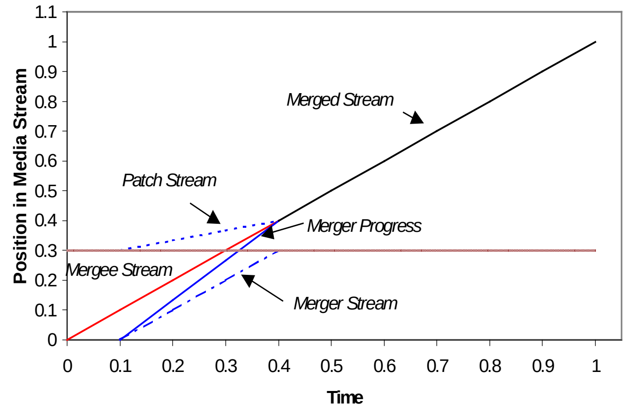

Figure 4: Operation of Latest Patch
(Only the merger stream listens to the patch stream.)

3.4 Shared Latest Patch

An obvious inefficiency of Latest Patch is that the data in the patching stream is sent twice during the merge operation: once to the merger and once to the mergee. Shared Latest Patch uses the bandwidth skimmed from the mergee clients to eliminate this overhead. That is, as illustrated in Figure 5 for a 25% skim, both the merger and mergee clients listen to the patch stream. When the mergee reaches the playback point corresponding to that data, its rate 1.0 stream is terminated and it continues playback out of its stored buffer (while continuing to receive data from the patch stream). This provides uninterrupted playback with only a single transmission of the data in the patch stream.

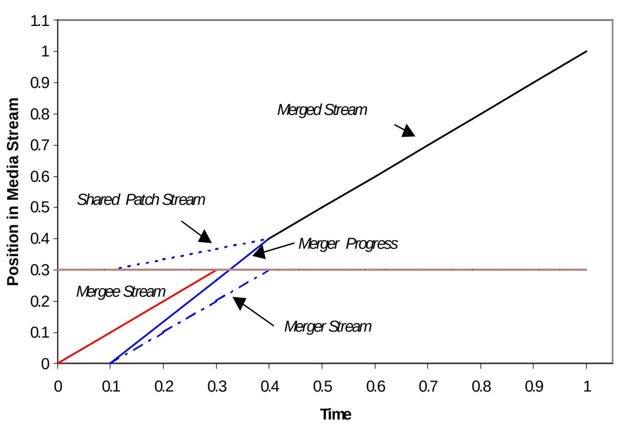

Figure 5: Operation of Shared Latest Patch
(*Both the merger and mergee streams listen to the patch stream.)*

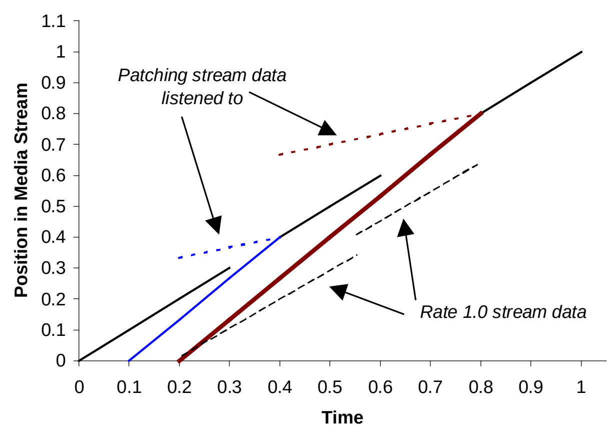

Figure 6: Discontinuous Patch
(Both the rate 1.0 and the patch stream received by the merger contain discontinuous media file data.)

The attraction of Shared Latest Patch is that the bandwidth consumed during the merge operation is reduced, since the rate 1.0 mergee stream can be turned off before the merge is complete. Thus, Shared Latest Patch is preferable to Latest Patch when a single merge operation is considered. However, when we consider the hierarchical merging tree created by the arrival of many client requests, Shared Latest Patch introduces two key inefficiencies. First, the mergee clients cannot listen to two different shared patches simultaneously, as is required if the mergee is simultaneously the target stream for two different merger streams. Second, and perhaps more importantly, if the mergee is receiving the shared patch for merging with the merger, then the mergee cannot simultaneously try to catch an earlier stream. Thus, a merge must be committed when a shared patch stream is initiated, which may result in poorer merge decisions. In Section 4, we will examine the maximum possible performance gain of Shared Latest Patch over Latest Patch using the precomputed optimal offline merging tree, with a deferred shared patch when a mergee is the merge target for two later streams. Shared Latest Patch will only be considered further if its performance for the offline optimal merge tree is substantially better than the performance of Latest Patch.

3.5 Discontinuous Patch

The Discontinuous Patch policy eliminates the problem that occurs when a mergee is the target of two different merger streams in the Shared Latest Patch policy. The basic idea is that the clients of the later merger stream simply listen to the shared patch stream already in progress. When the first merge completes, a new patch stream is created, and continues until the second merge operation finishes.

To illustrate the discontinuous transmission streams, Figure 6 shows the data transmitted when a second stream starts a merge with the same target stream as a merge that is already in progress. The thick line represents the rate of progress of the newest client, who arrives at time 0.2, and the dashed lines represent the (discontinuous) rate 1.0 merger stream for that client. The first shared patching stream that it listens to was initiated by the other client who arrived at time 0.1. (Patching stream data used by the other client alone has been removed to help clarify the figure). The Y-axis intervals covered by the dashed and dotted lines show the portions of the media file transmitted in the respective streams. It is easy to verify that all data required to effect the merge is received, and that each individual item is received in time to meet its viewing deadline.

A drawback of this approach is that the data received on the rate 1.0 and patching streams is not continuous in the media file. In addition to this implementation complexity, the Discontinuous Patch policy has not solved the second problem in the Shared Latest Patch policy, and thus will only be considered further if its performance in offline optimal hierarchical merging is significantly better than the Latest Patch policy.

4. OFFLINE POLICY COMPARISON

In this section we compare the proposed policies using approaches that isolate the distinctions among the policies' operations from the influence of choices of how to construct the full hierarchical merge tree in the presence of unpredictable client request arrivals. We refer to the analyses done here as offline, because they assume that full knowledge of future client request arrival times is available. Section 5 evaluates bandwidth skimming when subjected to online decision making, as would be the case in an actual implementation.

4.1 Merge Bandwidth Requirements

One key characteristic of a merging policy is the bandwidth that is consumed to merge two streams that are initially separated by a fixed distance t. Careful examination of the patching family and Partition policies reveals that, in some policies, some media file data is sent twice during a merge. This naturally raises the question of how efficient any conceivable policy can be in completing a merge – is there substantial room for improvement?

A simple lower bound on the bandwidth cost of a single merge operation can be derived from examination of Figure 2. Here a total interval of the media file data equal to at least t (for the merger stream) + $t(1\text{-}S)/S$ (for the mergee stream) = t/S must be transmitted to complete the merge of two streams separated by distance t. Thus, no merging policy can complete a merge at bandwidth cost less than t/S.

In contrast, it is easy to show that the Patch policies that complete a single merge most efficiently (Shared Latest Patch and Discontinuous Patch) require bandwidth $2(1–S)(t/S)$,† while Partition requires bandwidth $(1.5–S)(t/S)$, and Latest Patch requires bandwidth $(2\text{-}S)(t/S)$. These bandwidth requirements for a single merge are presented in Figure 7, normalized by the lower bound described above. We do not believe that the lower bound is achievable over the full range of S considered, although we have not proven this to be the case.

4.2 Policy Comparisons for Optimal Merge Trees

Figure 8 shows, for each of the proposed policies, the average server bandwidth consumed per client request, for a given media file, as a function of client bandwidth skim. The results are for the case that the request rate for the media file is an

† This expression assumes that $S \leq 0.5$. The expression for the full range of S is simply $\mathrm{Max}(t/S, 2(1\text{-}S)(t/S))$. We omit consideration of S > 0.5 in what follows to simplify the presentation.

average of 100 requests during the time required for a single full playback. (For a 2 hour movie, this would correspond to an average of one request every 1.2 minutes.) For each value of client bandwidth skim, we report average server bandwidth consumed per client in units of the playback rate for that given skim. Thus, the server bandwidth reductions shown in the figure are due only to client stream sharing; the additional reduction due to delivering poorer quality streams can be computed directly from the skim value. Note that when no client bandwidth is skimmed, a separate transmission stream is used for each client, and thus the server bandwidth per client is one.

The values are obtained using a combination of simulation and dynamic programming. The simulation is used to generate random sets of client request times. Each such set is fed to a set of dynamic programs, one per policy. The dynamic programs, which are relatively straightforward adaptations of those used in previous studies[1,10], each find the sequence of client merges (i.e., the merge tree) that minimizes the total cost of delivering the media data to all clients for that particular set of request times using the given policy. Enough trials were run to bring the confidence interval to within 5% of the mean. A dynamic program that uses merge costs given by the lower bound explained in Section 4.1 is used to provide an (almost certainly unachievable) lower bound on all possible policies.

The results show that the different tradeoffs made between Latest Patch and Shared Latest Patch have negligible effect on performance. Additionally, the added complexity of Discontinuous Patch brings no benefit for relatively small skims. Although Discontinuous Patch provides a discernable performance improvement over Latest Patch as the skim increases, the improvement is not substantial enough to argue for using this complex policy in an online implementation, where it will require premature merge commitments, as discussed in Section 3.4.

The results in the figure suggest that Partition is the technique of choice if the number of multicast channels required, the cost of more frequent changes in subscription to multicast channels, and the granularity of the skim options can be tolerated. If not, Latest Patch may be the best choice due to its ease of implementation and ability to defer commitments to merge until the merge completion points.

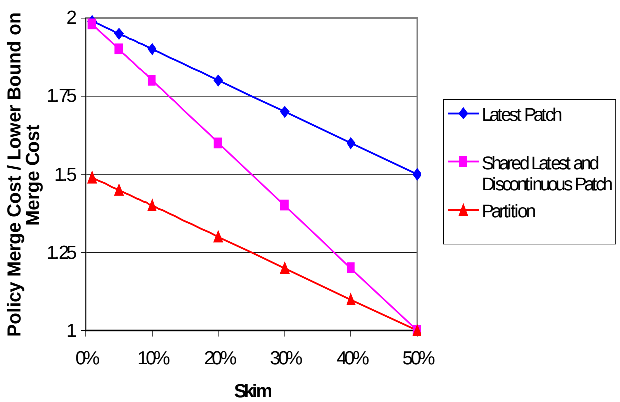

Figure 7: Bandwidth Required for a Single Merge Operation
(Values given are the ratios of the bandwidth consumed by each policy to the lower bound.)

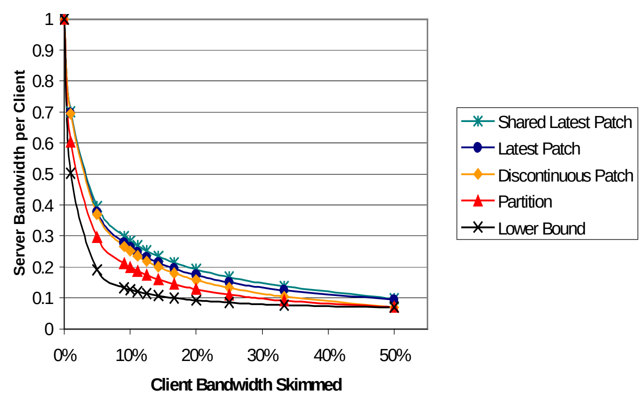

Figure 8: Server Bandwidth Using Optimal Merge Tree
(Bandwidth = 1 corresponds to devoting a server stream to each individual client request for the full duration of the playback)

5. PERFORMANCE EVALUATION OF CLIENT BANDWIDTH SKIMMING

In this section we use simulation to evaluate the performance of the Latest Patch and optimized Partition policies, in a setting with unpredictable client request arrivals, over a wide range of the system design space. The key question addressed is how much reduction in server bandwidth is possible, as a function of file popularity and the client bandwidth skim, in this online setting?

In each simulation, client request arrivals are generated according to a Poisson process or using a heavy-tailed distribution of interrequest times modeled by the Pareto distribution.‡ The “Earliest Reachable Merge Target” (ERMT) heuristic, defined in Section 2, is used to determine which stream merges should be attempted, and the optimized Partition or Latest Patch policy is used to effect the attempted merge. The relative performance of these online policies is similar to the relative performance of the offline versions of Latest Patch and Partition, as illustrated in Figures 8 and 9(a) for client request rate, *N*, equal to 100. Thus, the results are presented for the Partition policy; the results for the Latest Patch policy are very similar.

‡ We do not claim that the Pareto distribution is likely to be a good model of the client interarrival times for video-on-demand applications. On the contrary, it’s more likely that the arrivals will be Poisson [PaFl95], perhaps with a time-varying rate. However, we are interested in evaluating how robust our conclusions are to changes in the statistical nature of the client request arrival process.

As in the offline policy comparisons, the client arrival rate (*N*) is expressed as the average number of requests that arrive during the time required for a single client to view the object, and server bandwidth is measured in units of the playback rate, which is in turn determined by the size of the client bandwidth skim. All simulation results presented in this section, with the exception of those in Figure 9(b), have 95% confidence intervals that are within 5% of the reported values.[§]

5.1 Overall Benefit of Client Bandwidth Skimming

This section examines the overall reductions in delivery cost that are achieved by bandwidth skimming (specifically, with the Partition policy). These simulations assume that each client request is served immediately and that clients have enough local storage to buffer data that is received ahead of schedule.

Each curve in Figure 9(a) or (b) shows, for a given client request arrival rate, *N*, how the average server bandwidth consumed per client varies with the fraction of client bandwidth that is skimmed for stream merging. Figure 9(a) provides results for Poisson arrivals. Figure 9(b) contains results for the Pareto distribution of client request interarrival times, which are provided to illustrate the impact of alternative statistical assumptions regarding the client arrival process. Figure 10 shows a different view of the Poisson arrival results in Figure 9(a). The key observations from these figures are:

1. For files that are at least moderately popular (i.e., $N \geq 10$), skimming a small fraction of the client bandwidth (e.g., 15%) results in a sizeable reduction in delivery cost. Furthermore, for a given (small) skim, the benefit increases as the client arrival rate increases, as intuition would suggest.
2. Increasing the fraction of client bandwidth that is skimmed results in diminishing incremental benefit in terms of delivery cost reduction. For files with client request arrival rate greater than ten, skimming 16.7% of the client bandwidth achieves much of the benefit that is achievable when 50% of the client bandwidth is skimmed (i.e., when client reception bandwidth is twice the bit rate needed for viewing). (Prior work has shown that for the request arrival rates in the figures, the required server bandwidth with a 50% skim is within a factor of 1.65 of the minimum achievable when unlimited client bandwidth can be used to snoop on data transmissions.[9])
3. For files that are very popular (i.e., $N \geq 100$), the absolute decrease in per-client delivery cost is small when the skim is increased above 10%. However, the relative decrease in per-client delivery cost or the cumulative impact on total server bandwidth required may justify a larger skim, depending on the nature of the tradeoff against playback quality.

In general, skimming yields a bigger benefit, and a smaller skim yields a somewhat bigger relative benefit for the (more bursty) heavy tailed request interarrival times than for the Poisson request arrivals. However, the basic observations and design principles noted above appear to hold for both Poisson and Pareto arrivals, suggesting that our conclusions may be fairly robust for a range of client workloads. All subsequent results in this paper assume Poisson arrivals.

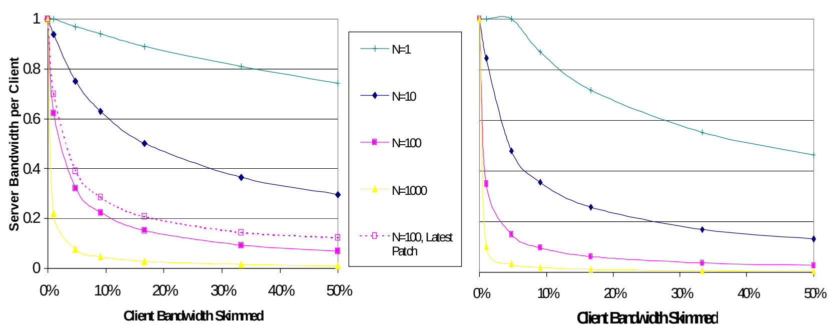

Figure 9: Effectiveness of Bandwidth Skimming
(Poisson on left, Pareto on right)

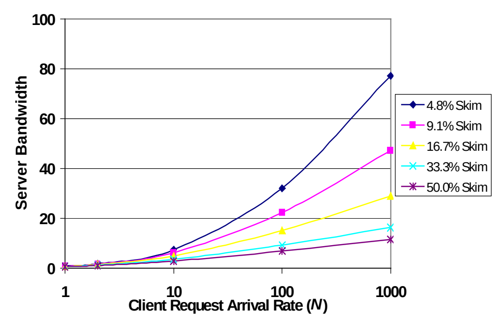

Figure 10: Total Server Bandwidth
(Poisson arrivals)

5.2 Impact of Limited Client Storage Capability

The results in Section 5.1 are derived from simulations that assume each client has sufficient local storage to buffer data that must be received ahead of its scheduled playback time, which enables all merges in which the client may participate. In this section we consider the impact of more restrictive client storage capabilities on the server bandwidth reductions that are achieved by bandwidth skimming. (Note that no stream merging is possible if absolutely no client storage is available.)

[§] Results for Pareto interarrival times are obtained by running each simulation for a single sample path of one million client requests, with the parameter α=1, and the parameter *k* chosen to achieve the desired average request rate as measured over the duration of the simulation.

The maximum storage required for buffering data decreases as the client bandwidth that is skimmed for stream merging decreases. Specifically, consider the total size of the highest quality file the client can receive (i.e., the total amount of data that is delivered when no bandwidth is skimmed). As a fraction of this total size, the maximum buffer use is given by $S(1\text{-}S)$. For a 9.1% ($1/11^{th}$) skim, for example, the maximum buffer requirement is slightly more than 8% of the "full rate" file.

Bandwidth skimming policies can be modified to handle clients that have local storage capacity less than the maximum by simply not performing merges that require buffer space that exceeds a client's available local storage. Figure 11 shows the delivery cost for the Partition policy as a function of client storage capability, for a 9.1% skim. Results for other skimming percentages are similar. The results show that if clients have local storage equal to about 50% of the maximum that is useful for merging (i.e., 4% of the full rate file when the skim is 9.1%), most of the benefit of bandwidth skimming can be achieved. For a 16% skim (results not shown here), local buffer space equal to 5-7% of the full rate file is sufficient.

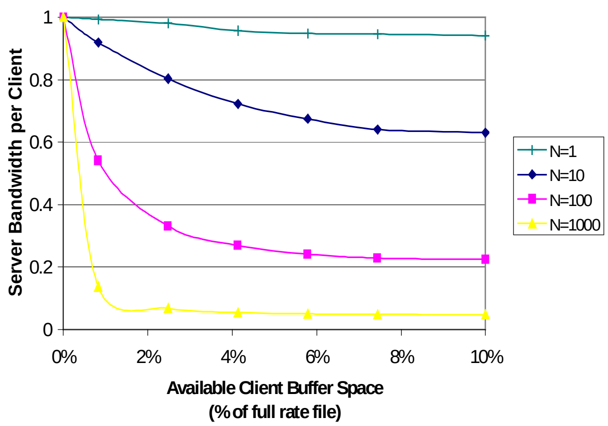

Figure 11: Performance versus Client Storage Capability *(9.1% skim)*

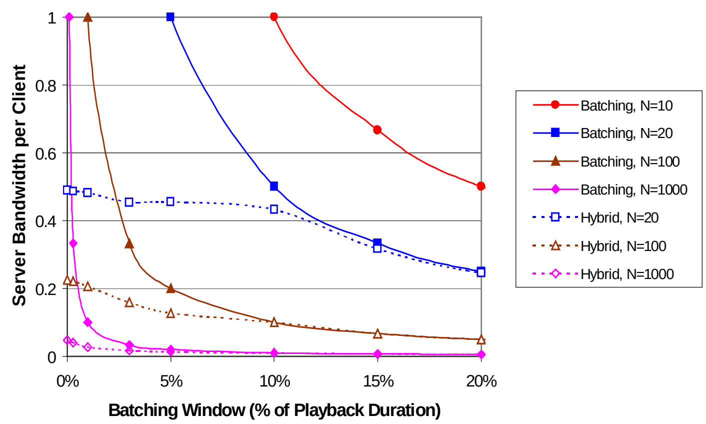

Figure 12: Performance of Simple Batching and Hybrid Batching/Bandwidth Skimming *(9.1% skim)*

5.3 Comparison of Bandwidth Skimming and Batching

In this section we compare bandwidth skimming to the previously proposed simple batching technique.[6] This batching technique requires no client bandwidth skim. Instead, client requests that arrive during a fixed-size window of time are batched together and served with a single multicast transmission that delivers the entire media file. This introduces latency in responding to client requests, or requires that an initial portion of the media file equal to the length of the batching window (e.g., 10% of the media file data) is pre-loaded in the client. A key advantage of batching is that, for any given batching window, the required total server bandwidth is independent of the client arrival rate.

We investigate whether there are regions of the system design space for which the batching policy with a relatively small batching window might be more effective than the bandwidth skimming policies. We also consider the server bandwidth reduction that is achieved by hybrid policies that batch client requests that arrive within a batching window and also use a small client bandwidth skim to merge the batched delivery streams.

Figure 12 shows the server bandwidth required per client as a function of the batching window size, measured as a fraction of the total time to view the media file, for both the simple batching technique (solid curves) and the hybrid batching / optimized Partition delivery technique (dotted curves). The hybrid technique uses a 9.1% client bandwidth skim. Each curve is for a different file popularity, as reflected in the file request arrival rate, N. Note that for batching window size equal to zero, the hybrid technique is identical to pure bandwidth skimming (i.e., Partition). The principal results in Figure 12 are:

1. In cases where simple batching achieves dramatic reductions in delivery cost, the hybrid batching/bandwidth skimming technique can be used to achieve the same or similar cost reduction using a much smaller batching window.
2. Compared with the pure bandwidth skimming technique, the hybrid technique with a batching window equal to 1-5% provides a decrease in delivery cost that may be significant for very popular files (i.e., $N \geq 100$).

6. CONCLUSIONS

This paper has addressed the problem of providing cost-effective video-on-demand (or more generally, on demand delivery of large data files that must be streamed to each client from beginning to end). The main contribution is a variety of techniques for "bandwidth skimming": using a small fraction of the client receive bandwidth, sufficiently small that display quality is not overly compromised, to achieve near-optimal hierarchical stream merging, and thus to reap substantial savings in server and network bandwidth. The results show that small skims (on the order of 15%) can result in large reductions in the server and network bandwidth required to provide immediate service to each client.

The proposed techniques vary in complexity and performance. Generally, it appears that the most important characteristic of such techniques with respect to the performance achieved is whether the online “early merging” heuristic can be used to merge streams as quickly as possible, in the presence of unpredictable client request arrivals. The bandwidth cost of effecting a merge also has an impact on relative bandwidth skimming policy performance.

On-going research includes (1) prototyping the Partition and Latest Patch bandwidth skimming techniques to investigate implementation costs and to design the client-server API for the techniques, (2) designing bandwidth skimming near-optimal hierarchical stream merging techniques that take into account the special characteristics of data that is encoded for compression and error recovery, and (3) developing optimized proxy caching strategies for systems that use the near-optimal hierarchical stream merging delivery techniques.

REFERENCES

1. C. C. Aggarwal, J. L. Wolf, and P.S. Yu, “On Optimal Piggyback Merging Policies for Video-On-Demand Systems”, *ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, Philadelphia, May 1996, pp. 200-209.
2. C. C. Aggarwal, J. L. Wolf, and P.S. Yu, “A Permutation Based Pyramid Broadcasting Scheme for Video-On-Demand Systems”, *Proc. IEEE Int’l. Conf. On Multimedia Computing and Systems (ICMCS’96)*, Hiroshima, Japan, June 1996.
3. Y. Cai, K. A. Hua, and K. Vu, “Optimizing Patching Performance”, *Proc. IS&T/SPIE Conference on Multimedia Computing and Networking 1999 (MMCN’99)*, San Jose, CA, Jan. 1999, pp. 204-215.
4. Y. Cai and K. A. Hua, “An Efficient Bandwidth-Sharing Technique for True Video On Demand Systems”, *Proc. 7th ACM Int’l. Multimedia Conference (ACM MULTIMEDIA ’99)*, Orlando, FL, Nov. 1999, pp. 211-214.
5. S. W. Carter and D. D. E. Long, “Improving Video-on-Demand Server Efficiency Through Stream Tapping”, *Proc. 6th Int’l. Conf. on Computer Communications and Networks (ICCCN’97)*, Las Vegas, NV, Sept. 1997, pp. 200-207.
6. A. Dan, D. Sitaram, and P. Shahabuddin, “Scheduling Policies for an On-Demand Video Server with Batching”, *Proc. 2nd ACM Int’l. Multimedia Conference (ACM MULTIMEDIA ’94)*, San Francisco, CA, Oct. 1994, pp. 15-23.
7. D. L. Eager and M. K. Vernon, “Dynamic Skyscraper Broadcasts for Video-On-Demand”, *Proc. 4th Int’l. Workshop on Multimedia Information Systems (MIS '98)*, Istanbul, Turkey, Sept. 1998, pp. 18-32.
8. D. L. Eager, M. C. Ferris, and M. K. Vernon, “Optimized Regional Caching for On-Demand Data Delivery”, *Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking 1999 (MMCN’99)*, San Jose, CA, Jan. 1999, pp. 301-316.
9. D. L. Eager, M. K. Vernon, and J. Zahorjan, “Minimizing Bandwidth Requirements for On-Demand Data Delivery”, *Proc. 5th Int’l. Workshop on Multimedia Information Systems (MIS '99)*, Indian Wells, CA, Oct. 1999, pp. 80-87.
10. D. L. Eager, M. K. Vernon, and J. Zahorjan, “Optimal and Efficient Merging Schedules for Video-on-Demand Servers”, *Proc. 7th ACM Int’l. Multimedia Conference (ACM MULTIMEDIA ’99)*, Orlando, FL, Nov. 1999, pp. 199-202.
11. L. Gao and D. Towsley, “Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast”, *Proc. 1999 IEEE Int’l. Conf. On Multimedia Computing and Systems (ICMCS’99)*, Florence, Italy, June 1999.
12. L. Gao, Z.-L. Zhang, and D. Towsley, “Catching and Selective Catching: Efficient Latency Reduction Techniques for Delivering Continuous Multimedia Streams”, *Proc. 7th ACM Int’l. Multimedia Conf.*, Orlando, Nov. 1999, pp. 203-206.
13. L. Golubchik, J. C. S. Lui, and R. Muntz, “Reducing I/O Demand in Video-On-Demand Storage Servers”, *Proc. ACM SIGMETRICS Joint Int’l. Conf. on Measurement and Modeling of Computer Systems*, Ottawa, May 1995, pp. 25-36.
14. K.A. Hua and S. Sheu, “Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems”, *Proc. ACM SIGCOMM’97 Conf.*, Cannes, France, Sept. 1997, pp. 89-100.
15. K. A. Hua, Y. Cai, and S. Sheu, “Patching: A Multicast Technique for True Video-On-Demand Services”, *Proc. 6th ACM Int’l. Multimedia Conf. (ACM MULTIMEDIA ’98)*, Bristol, U.K., Sept. 1998, pp. 191-200.
16. S. W. Lau, J. C.-S. Lui, and L. Golubchik, “Merging Video Streams in a Multimedia Storage Server: Complexity and Heuristics”, *ACM Multimedia Systems Journal 6*, 1 (Jan. 1998), pp. 29-42.
17. J.-F. Paris, S. W. Carter, and D. D. E. Long, “A Hybrid Broadcasting Protocol for Video On Demand”, *Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking 1999 (MMCN’99)*, San Jose, CA, Jan. 1999, pp. 317-326.
18. J.-F. Paris, D. D. E. Long, and P. E. Mantey, “Zero-Delay Broadcasting Protocols for Video-On-Demand”, *Proc. 7th ACM Int’l. Multimedia Conf. (ACM MULTIMEDIA ’99)*, Orlando, FL, Nov. 1999, pp. 189-197.
19. V. Paxson and S. Floyd, “Wide-Area Traffic: The Failure of Poisson Modeling”, *IEEE/ACM Transactions on Networking 3*, 3 (June 1995), pp. 244-266.
20. 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, July 1999.
21. S. Viswanathan and T. Imielinski, “Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting”, *Multimedia Systems 4*, 4 (Aug. 1996), pp. 197-208.