Optimal Scheduling of Secondary Content for Aggregation in Video-on-Demand Systems[1]

Prithwish Basu[2] Ashok Narayanan[3] Wang Ke Thomas D.C. Little
Department of Electrical and Computer Engineering, Boston University
8 Saint Mary's St., Boston, MA 02215, USA
Azer Bestavros
Department of Computer Science, Boston University
111 Cummington St., Boston, MA 02215, USA

***Abstract* – We present and evaluate an optimal scheduling algorithm for inserting secondary content for improving resource utilization in VoD systems. The algorithm runs in polynomial time, and is optimal with respect to the total bandwidth usage over the merging interval. We present constraints on content insertion which make the overall QoS of the delivered stream acceptable, and show how our algorithm can satisfy these constraints. We discuss dynamic scenarios with user arrivals and interactions, and show by simulations that content insertion reduces the channel bandwidth requirement to almost half.**

***Keywords:* Video-on-demand, service aggregation, secondary content insertion, scheduling**

I. Introduction

Non-uniform popularities of movies can result in skewed user access patterns in VoD systems[5] . Several techniques exploit this principle to aggregate individual users and serve them in groups. These resource sharing schemes map multiple "logical" channels onto a smaller number of "physical" channels to perform *service aggregation*[5, 9, 2, 6, 8, 10].

Stream merging minimizes end-to-end network bandwidth requirements by bridging the temporal skew between streams carrying the same content. This can be done by adaptive piggybacking[6] (we call it rate adaptive merging) and by content insertion[8]. Rate adaptive merging of two streams can be achieved by accelerating the trailing stream towards the leading stream by about 7%[4] until both are at the same position in the program. At this time, all users on both streams can be served off the same stream using multicast.

Secondary content insertion is similar to rate adaptation, but at a much coarser granularity. Here, the temporal skew between two streams is bridged by inserting short segments of secondary content into the leading stream, to allow the trailer to catch up. In [8], content insertion is presented in server overload situations and is unconstrained. We propose to use this technique to actively aggregate streams during normal operation of a VoD system. Clearly, indiscriminate insertion of content may cause unacceptable degradation of the viewing experience for some users. We address this problem by introducing a number of QoS constraints which bound the amount of secondary content inserted into streams, and also *shape* the inserted content to make the entire package acceptable.

The aggregation process involves two steps:clustering and merging. A clustering algorithm is used to generate *clusters* of streams to be merged [3]. A cluster consists of a number of streams, each serving the same content, but skewed temporally with respect to each other. The channels in a cluster are then merged by selectively inserting secondary content.

In this paper, we deal with stream merging. We discuss *optimal* techniques for scheduling of secondary content under different constraints with the primary goal of minimizing total bandwidth used during merging. We refer to this as the "static snapshot" case because a snapshot of the stream positions is taken at the beginning of the merging period and no user interactions are allowed to take place during this period. We begin with a simplified version of the problem where the inter-stream spacings are multiples of time-intervals equivalent to a group of ads and present a dynamic programming (DP) algorithm of time complexity $\mathcal{O}(n^3)$ to solve the problem. By adapting our earlier DP algorithm, a more general version of the problem can be solved. We also outline certain heuristics for the harder, "dynamic" version of the problem where user arrivals and interactions[5] are allowed to occur during the merging process. Throughout this paper, the terms "advertisement", "ad(s)" and "secondary content" have been used interchangeably, and they refer to the same thing.

Secondary content can take the form of advertisements, short news flashes, weather information, stock updates, sports scores, or other items of interest. Advertisements also serve to directly defray the cost of content production and service. We believe that such a scheme would help the VoD service provider in earning extra revenue, and at the same time subsidize the cost of programming to subscribers who are willing to receive QoS-constrained secondary content. Some

[1] This work is supported in part by the NSF under grants No. NCR-9523958 and CCR-9706685.

[2] correspondence to: *pbasu@bu.edu*

[3] now at Cisco Systems, Chelmsford, MA, USA

[4] an acceptable limit according to an empirical study

[5] Fast-Forward, Rewind, Pause, Quit

subscribers may wish to receive premium service with no advertisements, or receive all the ads at the beginning of the movie (near VoD). Our algorithm supports these cases too, optimally. Furthermore, these techniques are not restricted to the commercial VoD scenario, but can be extended to video-over-IP streaming frameworks as well.

With the increasing popularity of streaming media over the Internet, user demand may frequently outstrip the resources available at popular streaming servers. Using secondary content insertion, the server can continue supporting the existing users while merging them dynamically, meanwhile trying to accommodate new users, who would otherwise have been blocked.

The main contribution of this paper is an optimal solution for the QoS-constrained content insertion problem. The use of content insertion for bridging large skews and rate adaptation for fine-tuning has been described in [8]. Content insertion and excision have been discussed in conjugation with dynamic buffer management for near-VoD systems by Tsai and Lee [10]. Optimal techniques for performing rate-adaptive merging or adaptive piggybacking have been discussed in [3, 1].

Section II describes the problem and the constraints in detail; Section III proposes a DP based solution for the problem; Section IV discusses simulation results; Section V concludes the paper.

II. Problem Formulation

We define an *ad schedule* as a sequence of tuples, of the form (a_j, v_j). This represents delivery of advertisements for time a_j, followed by delivery of video content for time v_j. Fig. 1 depicts a possible ad schedule for a single user. When generalized to a set of U users, the ad schedule becomes a matrix of tuples (a_{ij}, v_{ij}), where each tuple represents the j^{th} pair of ad and video time given to user i. Typically, the interval a_{ij} will consist of a burst of multiple ads.

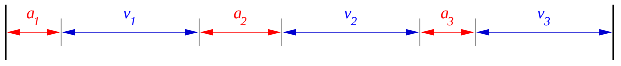


Fig. 1. Ad schedule for a single user

If the system could insert ads in an unconstrained manner, the optimal way to merge two users temporally separated by T seconds, is to keep the leader on ads for T seconds, and allow the trailer to catch up in this time period. For large values of T, this unacceptably degrades the viewing experience for the leader. Therefore, aggressive use of advertisement scheduling can succeed only when it is controlled by a set of QoS constraints which ensure that the viewing experience is not intolerably degraded due to advertising: no burst of ads should be excessively long; neither should ad bursts be delivered too close to each other; no user should receive more than a certain amount of ad time over some viewing interval; partial ads cannot be displayed. These constraints can be formally stated as follows:

$$a_{ij} = nA_{min} \qquad n \in \mathcal{Z}^+ \quad \forall_{i,j} \tag{1}$$

$$a_{ij} \leq A_{max},\ v_{ij} \geq V_{min} \qquad \forall_{i,j} \tag{2}$$

$$\sum_T a_{ij} \leq \alpha \sum_T (a_{ij} + v_{ij}) \quad \forall_{i,j} \tag{3}$$

In this paper, we assume that all advertisements are of the same length, A_{min} which represents the granularity of every a_{ij}. However, this approach can easily be extended to serve ads of different lengths, as long as all the ad-lengths are integer multiples of some base value A_{min}. A_{max} is the maximum length of a single ad burst. Clearly, A_{max} should also be an integer multiple of A_{min}. V_{min} is the minimum video time that has to occur between two ad-bursts. There are two limits on the fraction of viewing time that can be used to display ads. One is the *long-term ad-dosage limit* (α, T), which represents the fraction of viewing time available for ads α, over some time interval T. This is a pinwheel scheduling constraint [7], applicable over any time interval T. The other is the *short-term ad-dosage limit* which represents the maximum rate at which ads can be inserted in video. It is given by $\beta = \frac{A_{max}}{A_{max}+V_{min}}$. It is easy to see that in general, $\alpha \leq \beta$.

The problem which we are trying to address in this paper is the following - "*For a group of N streams carrying the same content but at different points in time (i.e. if a snapshot of the stream positions is taken at a particular time instant), what is the ad-video schedule that minimizes the total bandwidth while merging them into one stream, at the same time obeying the above QoS constraints?*"

If at the start of the merging cycle, stream i was at position p_i and stream j was at position p_j, then for these two streams to merge, the following relation should hold: $p_i - p_j = nA_{min}$, $n \in \mathcal{Z}^+$. This implies that ad scheduling can only be used to bridge skews which are integral multiples of the minimum ad length. In practice, such temporal skews are uncommon, therefore ad insertion must be coupled with another, more fine-grained aggregation technique like rate adaptive merging [6].

III. Our Solution Approach

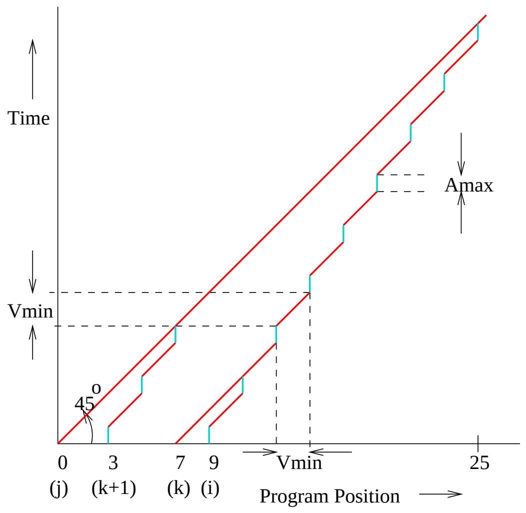
In this paper we discuss a restricted version of the problem and owing to paucity of space, direct the reader to [4] for the details regarding the general version of the problem. Following are the simplifying assumptions:

$$\alpha = \beta \tag{4}$$

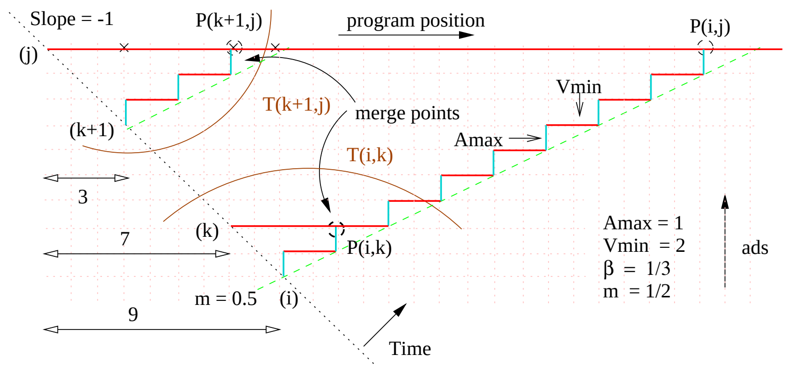
$$a_{ij} = 0\,|\,A_{max} \qquad \forall_{i,j} \tag{5}$$

$$v_{ij} = V_{min} \qquad \forall_{i,j} \tag{6}$$

In brief, we simplify the problem by making the two constraints on ad dosage equal. Also, we constrain the ad



(a) Intuitive Representation



(b) Rectilinear Representation

Fig. 2. Restricted ad schedule grid for multiple users

dosage to zero, or a fixed value which is the maximum we can give. Inter-ad video dosage is also fixed at the minimum possible limit. It is clear that any solution satisfying these constraints will also satisfy the general constraints presented earlier. We also impose the following additional constraint on program positions at the beginning of the snapshot, so that only programs whose difference in positions is an integral multiple of our ad dosage unit, can be merged.

A. Preliminaries

We first introduce a graphical notation to denote the mergeable streams with different content progression rates on a time scale. Fig. 2(a) shows an intuitive representation of the streams on Cartesian axes; diagonal motion along a line with slope 1 refers to a stream with normal speed and vertical motion refers to a stream in content-insertion state. In the particular example, there are 4 streams receiving the same program but their positions are skewed in program time as indicated in the figure. Our task is to alter the content progression rates of these given streams from the initial time instant and merge them into one stream. This merging process should consume minimum network bandwidth.

For simplicity, we use a rectilinear representation of this graph, as shown in Fig. 2(b). The figure has two time axes: horizontal for video time, and vertical for ad time. Units on both these axes are taken to be equal. Note that for two streams with an initial skew Δ to merge, the leading stream must be given ads for Δ time more than the trailing stream, for any Δ. The leading stream is farthest to the right (Stream i in the figure) and the trailing stream is the one at the top (stream j). Other streams are placed along the horizontal time axis according to the skews shown in Fig. 2(a). These are denoted by $\times$ marks on the horizontal line at the top of Fig. 2(b). Then, we can plot a line with slope -1 through the initial point, and project the initial temporal skews from the horizontal (video) time-line onto the diagonal to obtain the initial points on the graph. In this rectilinear grid graph, a horizontal step represents video delivery for time V_{min} on that stream, and a vertical step represents ad delivery for time A_{max}. At any given point in time, a diagonal line with slope -1 gives the locus of all stream positions. We can visualize this diagonal line sweeping towards the North-East direction across the grid as time progresses. A *merge point* is defined as a point where two or more streams merge into one. A *segment* is a section of a stream between two merge points, or between the start point and the first merge point. We can see that a merging schedule in Fig. 2(a) becomes a staircase like pattern in Fig. 2(b). The terms $T(\cdot)$ and $P(\cdot)$ have been defined later in Section B.

Our solution relies on the following observations. Detailed proofs of these are presented separately in [4].

Observation 1 *To achieve optimality, at any time a segment can only be in one of the two states: decelerated and steady. A segment in decelerated state receives the maximum ad-dosage available. A segment in steady state receives no ads. Also, at each merge point, exactly two segments merge into a single segment. These can be deduced by noting that a segment lies between merge points, and therefore does not contain any merge points within itself. Therefore, the aim of giving ads to a stream can only be to slow it down so that a trailing stream may catch up. Clearly the optimal merging policy will decelerate the segment at the maximum possible rate so that the merge happens earlier and a channel is released sooner. This has been illustrated in Fig. 3. It is clear that both case (b) and case (c) have less cost in terms of bandwidth usage than case (a), since the merge happens later in (a).*

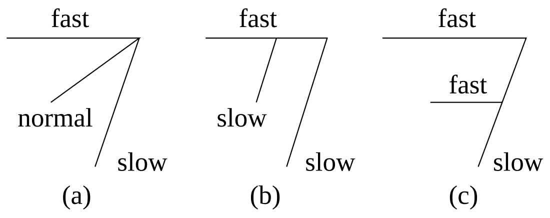


Fig. 3. Merging of three streams

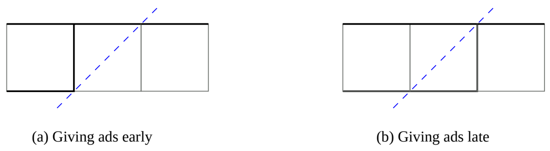


Fig. 4. Ad dosage earlier vs. later

Observation 2 *If advertisements are to be inserted in a segment, it is less costly to give advertisements as early as possible, and video content later. In Fig. 4(a), the decelerated stream merges earlier with the trailing stream than in 4(b), because ads are given earlier, hence the bandwidth consumed in the former case is less than in the latter.*

Observation 3 *For a decelerated segment, the optimal ad scheduling technique is to give the maximum possible ad dosage (A_{max} in this case) in the beginning, followed by the video complement for this ad dosage. This pattern is repeated periodically throughout the segment. This follows from the previous observation and the pinwheel scheduling constraint in (3).*

Observation 4 *The point where all streams finally merge occurs at a time V_{min} before the intersection of a horizontal line corresponding to the trailing stream, and a diagonal line of slope $m = \frac{\beta}{1-\beta}$ drawn through the initial point (projected on the line with slope -1) of the leading stream. This line has been shown in Fig. 2(b). All streams are constrained within the envelope formed by these two lines because these two lines depict the maximum and minimum ad dosage, therefore all streams must receive ad dosage between these.*

In Fig. 2, the ad constraint envelope is shown by a diagonal line of slope m. For convenience, m is shown here to be $\frac{1}{2}$; in practice, m would be considerably less (around $\frac{1}{6}$, which translates to a maximum of 10 minutes of ads per hour of viewing time).

Theorem 1 *The graph with segments formed from a merging schedule for a given scenario is a binary tree where the average slope of each segment is either 0 (no ads) , or $m = \frac{\beta}{1-\beta} = \frac{A_{max}}{V_{min}}$ (ad-video bunches). Also, finding the optimal (minimum bandwidth cost) merging schedule is isomorphic to finding the optimal binary tree on n leaf nodes, separated by given distances.*

This follows from the previous observations. See [1, 3] for similar results on rate-adaptive merging.

B. *Solution to the Restricted Case*

Since the number of binary trees with n leaf nodes grows exponentially with n, exhaustive search of all possible trees is impractical for any large value of n. However, a dynamic programming approach helps us to solve this problem in polynomial time. We outline the solution below.

We number the streams from 1 to n, with 1 being the leading stream and n being the trailing stream. Let L be the length of the movie (last program position). Consider the two streams, i and j in Fig. 2(b), with $i < j$ and $p_i > p_j$. From observation 4, we can see that $P(i, j)$ is the optimal merge point for streams $i, \dots, j$ and is given by (7) and (8).

$$\begin{aligned} P(i,j) &= p_i, & i = j \qquad (7)\\ &= p_j + \frac{p_i - p_j}{\beta} - V_{min}, & i \neq j \qquad (8)\end{aligned}$$

Let $T(i, j)$ denote an optimal binary tree for merging streams $i, \dots, j$ and $C(i, j)$ denote the *cost* of this tree all the way to the end of the program. The cost of a binary tree is the sum of the lengths of all the segments in the tree. This is equivalent to the *total* bandwidth consumed until all given streams merge into one. Obviously, if $T(i, j)$ is optimal, then $C(i, j)$ is the minimum possible cost for merging streams $i, \dots, j$. Since this is a binary tree, there exists a point k such that the right subtree contains the nodes $i, \dots, k$ and the left subtree contains the nodes $k + 1, \dots, j$. From the "principle of optimality", which holds for binary trees, if $T(i, j)$ is optimal, then both the left and right subtrees must be optimal, i.e., $T(i, k)$ and $T(k + 1, j)$ must be optimal. The cost of this tree is given by a dynamic programming formulation ((9) and (10)), and the optimal policy merges $T(i, k^*)$ and $T(k^* + 1, j)$ into $T(i, j)$, where k^* is given by (11).

Here $C(i, k)$ and $C(k + 1, j)$ are the costs of the right and the left subtrees respectively, calculated all the way till the end of the movie. The third term is subtracted to eliminate the cost duplication after the streams i and j merge. The fourth term is added to include the ad time after $P(i, k)$ into the cost formulation. This is because, even if a certain stream has been put on ads momentarily, the resources allocated to it in the server and the network (in case of bandwidth reservation) cannot be freed until it actually merges with some other stream. Since the number of ad channels is assumed to be fixed (ideally, one multicast ad channel suffices), the bandwidth costs due to those channels do not feature in (10).

We begin by calculating $T(i, i)$ and $C(i, i)$ for all i. Then, we calculate $T(i, i + 1)$ and $C(i, i + 1)$, then $T(i, i + 2)$ and $C(i, i + 2)$ and so on, until we find $T(1, n)$ and $C(1, n)$. This gives us our optimal cost. The algorithm has been summarized below:

$$\begin{aligned} C(i,j) &= L - p_i, & i = j \quad (9)\\ &= C(i,k) + C(k+1,j) - \max(L - P(i,j), 0) + (p_k - p_j), & i \neq j \quad (10)\\ k^* &= \arg\min_{i \le k < j}\{C(i,k) + C(k+1,j) - \max(L - P(i,j), 0) + (p_k - p_j)\} & (11) \end{aligned}$$

```
Algorithm DP_Find_Tree
{
  for ($i$=1 to $n$)
    initialize $P(i,i)$, $C(i,i)$ and $T(i,i)$
    from (7) and (9)
  for ($p$=1 to $n-1$)
    for ($q$=1 to $n-p$)
      Compute $P(q,q+p)$, $C(q,q+p)$ and $T(q,q+p)$
      from (8), (10) and (11)
}
```

There are $\mathcal{O}(n)$ iterations of the outer loop and $\mathcal{O}(n)$ iterations of the inner loop. Additionally, determination of $C(i,j)$ requires $\mathcal{O}(n)$ comparisons in the arg min step. Hence, the algorithm DP_Find_Tree has a complexity of $\mathcal{O}(n^3)$. A point to be noted here is that in real systems, n is not likely to be very high, thus making the complexity acceptable. We show later in the simulation section that not much optimality is lost by reducing the size of a snapshot.

C. *The General Case*

We relax the constraint imposed by (5), making a_{ij} variable, while still being subject to (1) and (2). We also remove the constraint imposed by (4) resulting in the appearance of both the *short-term* and the *long-term* ad constraints, as defined in Section II. Like in the restricted situation, the optimal content insertion policy schedules as much ad time as early as possible, and then fills in the video as necessary. The calculation of $P(i,j)$ is more complicated in this case since a merge point can occur during an ad-burst. But the structure of the merging tree remains binary and hence `DP_Find_Tree` can still be applied with a modified expression for $P(i,j)$. Due to space limitations, we cannot include details of our optimal algorithm here, and direct the reader to a longer version of the paper[4].

The algorithms described till now generate optimal schedules only in the static snapshot case. But while designing real VoD systems, one needs to find techniques for handling user interactions without sacrificing optimality. A standard way of achieving this goal is to re-compute the optimal schedule by `DP_Find_Tree` periodically. This period can be based on time or the rate of arrivals/interactions. However, if the rate of interactions is high, then the re-computations have to be done very frequently. Another technique that sacrifices some optimality is a *segment fitting* technique, which maintains the original merging schedule and tries to optimally "connect" the (interactive) streams that have cropped up in the interior of the merging tree, to the original tree, if possible. Of course, this technique may lead to highly suboptimal solutions under specific situations and has no good upper bounds on the cost increase. We conduct simulations to observe the effectiveness of this segment fitting heuristic.

TABLE 1
SIMULATION PARAMETERS

Parameter	Meaning	Value
M	Number of movies[a]	100
L	Length of a movie	120 min
A_{min}	Length of one ad	30 sec
A_{max}	Max. length of an ad-burst	2 min
V_{min}	Min. video time between ad-bursts	8 min
T	Long term time window	60 min
α	Frac. of ad-time in the long run	$\frac{1}{6}$
R	Re-computation interval	1200 sec
B	Initial batching interval	$A_{min} = 30$ sec
λ_{arr}	Mean inter-arrival rate[b]	0.0833 s^{-1}
λ_{int}	Mean interaction rate[b]	0.07 s^{-1}
λ_{dur}	Mean interaction duration[b]	5 sec
f	Rate of Rewind/FF	5x

IV. SIMULATION RESULTS

In this section, we describe the simulation results of a general dynamic scenario where the users come into the system, interactively view the movie, and then leave[6]. Our primary performance measure is the ratio of the number of running streams to the number of users in the system, as it directly quantifies the gains due to secondary content insertion.

A re-computation period, R is an interval of time during which a content insertion algorithm attempts to release channels, and after which a new snapshot is taken. Initially all streams are run for time R at normal speed and only then the ad-insertion starts. At the beginning of every snapshot, algorithm `DP_Find_Tree` computes the *optimal* paths for each stream, and until the next snapshot happens, all currently running (non-interacting) streams follow the paths as prescribed by the algorithm. Newly arrived streams, however are allowed to run at normal speed until the next snapshot. One important assumption that we make in the interaction

[a]Zipfian popularity distribution
[b]Exponential distribution assumed
[6]Simulations of static scenarios have been described in [4].

model is that interactions are not allowed during ad-bursts. All interactions that occur during an ad-burst are serviced at the end of the burst. When a user interacts, he/she is allocated a new stream at least for the duration of interaction. But after resuming the user can be merged with any other stream since the ad-budget of that stream is reset after the interaction event. This is to discourage users from interacting for the sole purpose of skipping the ads.

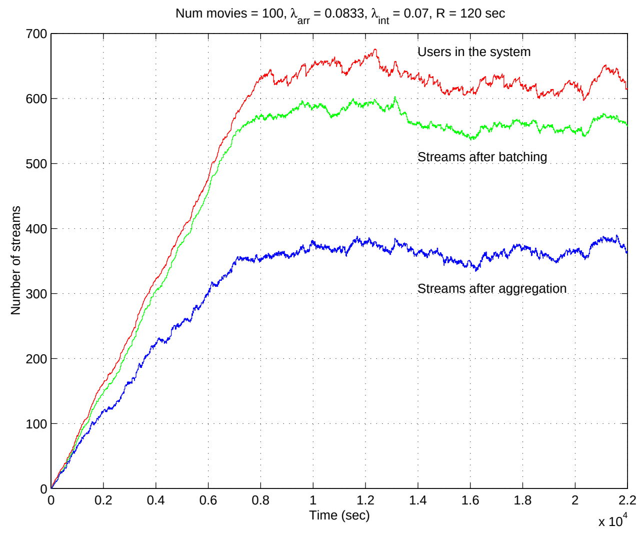


Fig. 5. Dynamic Case with Arrivals and Interactions

The simulation parameters have been listed in Table 1. We simulate the case where the most popular movie has arrivals once every minute, and it translates to the aggregate arrival rate of $\frac{1}{12}$ sec^{-1} for 100 movies. Fig. 5 shows the gains due to ad-insertion in this dynamic interactive scenario. The average number of users in the system U should approximately be $\lambda_{arr} \times L = 600$, in our case. The simulations show a slightly higher value (around 625) since ad-insertion slows users down resulting in more number of users. But, after aggregation, the number of streams in the system is only around 350, which directly translates to a 45% saving in capacity. On the other hand, batching reduces the channel bandwidth requirement by a mere 10%. Therefore secondary content insertion helps us in cutting down the bandwidth requirement to almost half the original amount; the spare bandwidth, if any can be used to serve a larger number of customers.

We also increased the re-computation interval from 120 seconds to 1200 seconds, and observed no appreciable increase in resource usage. Thus we concluded that for low to medium interaction levels, segment fitting heuristics work well and re-computation can be done relatively infrequently, hence reducing the computational overhead of the algorithm significantly.

V. Conclusions and Future Work

In this paper, we presented and evaluated an optimal algorithm for inserting secondary content for service aggregation in an interactive VoD system. We demonstrated that the algorithm runs in $O(n^3)$ time, where n is the number of streams in a cluster. We have presented the simulation results for a fully interactive scenario where the users are arriving, interacting and leaving the system. For a mean arrival rate of around $\frac{1}{12}$ sec^{-1}, and a combined interaction rate of around 0.07 sec^{-1}, we show almost 50% reduction in the number of channels required.

Personalization and value-added secondary content such as news are important factors in increasing the acceptability of this solution. Another important factor for the success of ad insertion is an appropriate pricing policy which provides enough subsidies to the user. Also, uniform ad insertion into a video stream is not always feasible due to the occurrence of “gripping” scenes in the movie. Metadata can be inserted off-line, into certain points of the stream, instructing the server not to attempt any aggregation at those points in time.

The problem of differentiated content insertion at multiple levels needs to be examined. Analyses of the effects of changing access patterns and interaction rates on the performance of our algorithm are currently underway.

References

[1] C.C. Aggarwal, J.L. Wolf and P.S. Yu, “On Optimal Piggyback Merging Policies for Video-on-Demand Systems,” *Proc. SIGMETRICS ’96: Conference on Measurement and Modeling of Computer Systems*, Philadelphia, PA, USA, pp. 200-209, May 1996.

[2] K.C. Almeroth and M.H. Ammar, “On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service,” *IEEE Journal on Selected Areas in Communication*, Vol. 14, No. 6, pp. 1110-1122, Aug 1996.

[3] P. Basu, R. Krishnan, T.D.C. Little, “Optimal Stream Clustering Problems in Video-on-Demand,” *Proc. Parallel and Distributed Computing and Systems ’98 - Special Session on Distributed Multimedia Computing*, Las Vegas, NV, USA, pp. 220-225, Oct 1998.

[4] P. Basu, A. Narayanan, W. Ke, T.D.C. Little and A. Bestavros, “Optimal Scheduling of Secondary Content for Aggregation in Video-on-Demand Systems,” *MCL Technical Report MCL-TR-12-16-98*, Dec 1998. URL: `http://hulk.bu.edu/pubs/papers/1999/basu-adins99/TR-12-16-98.ps.gz`

[5] A. Dan, D. Sitaram, and P. Shahabuddin, “Scheduling Policies for an On-Demand Video Server with Batching,” *Proc. ACM Multimedia*, San Francisco, CA, USA, pp. 15-23, Oct 1994.

[6] L. Golubchik, J.C.S. Lui and R.R. Muntz, “Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-On-Demand Storage Servers,” *Multimedia Systems*, ACM/Springer-Verlag, Vol. 4, pp. 140-155, 1996.

[7] R. Holte *et al.*, “The pinwheel: A real-time scheduling problem” *Proc. 22nd Hawaii International Conference on System Science* pp 693-702, Kailua-Kona, HI, USA, Jan 1989.

[8] R. Krishnan, D. Venkatesh and T.D.C. Little, “A Failure and Overload Tolerance Mechanism for Continuous Media Servers,” *Proc. Fifth Intl. ACM Multimedia Conference*, Seattle, WA, USA, pp. 131-142, Nov 1997.

[9] W.D. Sincoskie, “System Architecture for a Large Scale Video on Demand Service,” *Computer Networks and ISDN systems*, Vol. 22, pp. 155-162, 1991.

[10] W.-J. Tsai and S.-Y. Lee, “Dynamic Buffer Management for Near Video-On-Demand Systems,” *Multimedia Tools and Applications*, Kluwer Academic Publishers, Vol. 6 Issue 1, pp. 61-83, Jan 1998.