Server-Based Service Aggregation Schemes for Interactive Video-on-Demand[1]

P. Basu, T.D.C Little, W. Ke, and R. Krishnan

Department of Electrical and Computer Engineering
Boston University, Boston, Massachusetts 02215, USA
(617) 353-9877
{*pbasu,tdcl,ke,krash*}*@bu.edu*

MCL Technical Report No. 09-01-2002

Abstract– The support of independent VCR-like interactions in true video-on-demand systems is resource intensive and in the worst case requires a separate channel for each user. A variety of techniques have been proposed to reduce resource requirements by aggregating users into groups. Recent research efforts have focused on client based aggregation owing to the massive reduction in prices of memory and storage. However, with the evolution of new computing paradigms such as pervasive computing, in which streaming media are anticipated and portable clients do not have enough buffering capacity or good network connectivity, we believe that the importance of server-based aggregation schemes will increase. Since such schemes do not assume any end-client capacity requirements, they are the logical candidates for enabling efficient VoD service in heterogeneous pervasive computing environments.

Clustering of users by bridging the temporal skews between them is one such server based service aggregation technique. In this chapter we present a survey of recurrent problems in stream clustering and discuss optimal solutions with a goal of minimizing average bandwidth. We show that for dynamic interactive scenarios in

[1]In *Handbook of Video Databases: Design and Applications*, Borko Furht and Oge Marques, Eds, CRC Press, Boca Raton, FL, 2004, pp. 927-960. This work is supported by the NSF under grant No. NCR-9523958 . Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

which streams can break away from clusters, given perfect prediction, the problem is isomorphic to the Rectilinear Steiner Minimal Arborescence (RSMA) problem which has been recently proved to be NP-hard. Since the RSMA model relies on some unrealistic assumptions, we also review heuristic techniques, which are essentially periodic invocations of the statically optimal algorithm, and show results of their performance evaluation. We demonstrate that the heuristic algorithms perform reasonably well in dynamic situations. These algorithms are general across a large class of implementation architectures and aggregation techniques.

We also describe a modeling of an interactive VoD system using stochastic dynamic programming with an average cost formulation. However, the size of the state space makes the optimization problem impractical, and we resort to other variants of RSMA such as predictive RSMA, and periodic RSMA with the same frequency as that of the mean of aggregate event process. We show that the former does not improve performance by much, and the latter performs better than other variants of the algorithm.

Keywords: Interactive video-on-demand, dynamic service aggregation, stream clustering, dynamic programming.

1 Introduction

Interactive video-on-demand (i-VoD) describes one of the most attractive applications that have been proposed since the rapid advances in high speed networking and digital video technology. Principal deterrents to the widespread deployment of i-VoD services are the cost of data storage server and network capacity to the end user. Because of the data-intensive and streaming nature of video data, fully provisioned interactive channels are prohibitively expensive to deploy. Service aggregation schemes attempt to alleviate these problems by sharing both server and network resources at the expense of small overheads in terms of delay and/or buffer space. Many techniques exist in the VoD literature (although some of them only support non-interactive VoD) – Batching [6], server caching [24, 7], client caching or bridging [1] and chaining [22] (a limited form of distributed caching), rate adaptive merging [11], content insertion [17, 25], content excision [25], periodic broadcasting [14], and patching [13, 8], to name a few.

A major bottleneck in the delivery of video is on the disk-to-server path. Interval-Caching solutions [7] attempt to minimize the bandwidth required on this path by caching intervals of video between successive disk accesses and then by serving subsequent requests from the cache. These schemes do not however, attempt to minimize the network bandwidth on the subsequent server-to-client path.

Clustering schemes, in contrast, seek to minimize this network bandwidth by aggregating channels that are identical except for their potentially disparate start times. That is, they are "aggregatable" if their skew is within some bound, or their skew can be eliminated[2]. Batching is a clustering technique in the near-VoD scenario, in which users must wait until the next launch of a stream occurs. A more attractive technique for the end user is a technique called rate adaptive merging [11, 16] in which a user acquires a stream without any waiting time; however, the system attempts to aggregate streams within a skew bound by *minimally* altering

[2] *Skew* is defined as the difference in program positions at current instant of time of two video streams containing the same program.

the "content progression rates" of a subset of these streams. Experiments have shown that speeding up a video stream by approximately 7% does not result in a perceivable difference in visual quality [17, 4], giving credibility to this approach. A trailing stream, or streams, can be accelerated towards a "leading" stream, or streams, until both of them are at the same position in the program (or equivalently, their skews are zero) [16]. At this instant of zero skew, all users attached to the stream (e.g., via multicasting to multiple destinations and users) are served by a single stream from the video source.

Both batching and rate adaptive merging have the advantage of having the burden of implementation fall mainly on the server and partly on the network (through the ability to support multicast), with no extra requirement on the client except that it be able to display the data received and send back control signals (to make requests and perform VCR-like operations). This means that potentially "true-VoD" service can be extended even to wireless handheld devices that do not have a large storage capacity but which are connected to the network. Client based aggregation schemes which require clients to buffer video from multiple staggered broadcast channels [14, 13, 8] are thus unsuitable for heterogeneous pervasive computing environments.

It is also desirable to reduce the buffering requirement for interval caching schemes. This requirement is dependent on the skews between streams carrying the same content. Therefore, bridging the skew and/or reducing the number of streams by clustering can also reduce the memory requirement of a caching scheme. A continuum of aggregation schemes which lie between pure caching and pure clustering exist. Tsai and Lee [25] explore this relationship in the near-VoD scenario. Therefore, one can extend solutions for clustering to buffer management and vice versa. In this chapter, we focus on *clustering* schemes.

In hybrid CATV architectures that consist of the standard broadcast delivery mechanism augmented with on-demand channels, stream merging is used in schemes like *Patching* [8] in order to reclaim the on-demand channels. The described clustering algorithms are applicable in such scenarios as well.

The scope for aggregation of video streams is ameliorated by the long-lived nature of video traffic and the high density of accesses on popular titles (e.g., hit movies) during certain times of the day. By clustering streams during peak periods, a system can accommodate a much larger number of users as than without aggregation, thus increasing potential revenue earned. Therefore, in any VoD system where peak demands are higher than the available bandwidth, aggregation schemes are attractive. This is particularly true of scenarios that commonly arise with Internet traffic and content. For example, when a popular news item is made available by a news provider, the peak demand can far outstrip the available network bandwidth, thus making aggregation increasingly attractive.

Because resource sharing by aggregating users is a promising paradigm for VoD systems supporting large user populations, it is useful to study its inherent complexity. Irrespective of whether we are trying to use rate adaptive merging or content insertion or using "shrinking" to reduce buffer usage for caching, the underlying clustering problem remains the same. In this chapter, we take a systematic approach to formulating these clustering problems and discuss the existence of optimal solutions. These algorithms are general across a large class of implementation architectures as well as aggregation techniques.

From a performance engineering standpoint, one must not lose sight of the fact that simpler solutions while provably sub-optimal can at times be preferable due to simplicity, elegance or speed. Often, they can be necessitated by the fact that optimal solutions are computationally expensive or intractable. In such cases, the engineering approach is to seek good approximate or heuristic alternatives that provide near-optimal solutions. With this in mind, we present optimal solution approaches and also investigate when such sophisticated algorithms are warranted by the application under consideration. In the "static" version of the problem, clusters of users receiving the same video content are formed but it is assumed that the users cannot break away after their streams have been merged with others in a cluster. If we allow the users to break away from their clusters by interacting, the gains due to aggregation will be lost. Hence, dynamic approaches are necessary for handling these interactive

situations. We present some such approaches and also explore how far the gains from an optimal solution in the static case are retained in dynamic scenarios. Our performance results can be readily applied to the related capacity planning and design problem, that is, given the mean arrival (and interaction) rate and distribution, what should the design capacity be, in number of streams, of a video-server network which actively uses service aggregation?

We transform the static clustering problem (with the objective of minimizing average bandwidth) to the RSMA-slide problem (terminology used by Rao et al. [21]). An important observation here is that an RSMA-slide on n sinks is isomorphic to an optimal binary tree on n leaf nodes. Aggarwal et al. have described an optimal binary tree formulation in reference [2], but we describe a more general approach that can model additional interactive scenarios. By periodically recomputing the RSMA-slide with small a period, we can reclaim up to 33% of channel bandwidth when there are $1,250$ interacting users in the system watching 100 programs, in steady state. We compare the RSMA-slide algorithm with other heuristic approaches and find it to be the best overall policy for varying arrival and interaction rates of users. We also find that under server overload situations, EMCL-RSMA (forming maximal clusters followed by merging by RSMA-slide in the clusters) is the best policy as it releases the maximum number of channels in a given time budget, consuming minimum bandwidth during the process. We also investigate some variants of the RSMA-slide algorithm such as event-triggered RSMA, Stochastic RSMA etc. and compare their performance with the RSMA-slide.

Related Work The static stream clustering problem has been studied by Aggarwal et al. [2] and Lau et al. [18]. The latter provides a heuristic solution for the problem while the former discusses an optimal solution based on binary trees. Neither discuss how to accommodate user interactions although Aggarwal et al. consider implications of user arrivals. Basu et al. investigate the stream clustering problem from a user arrival/interaction standpoint with early performance results [3].

An important contribution of this work is the inclusion of user arrivals and interactions into the stream clustering model and showing that iterative application of optimal static algorithms in a controlled fashion can preserve most of the gains in dynamic interactive situations too. We use a different problem formulation that allows us to model most interactive situations although in the static case it reduces to the optimal binary tree formulation in [2]. We point out that the problem of freeing maximum number of channels under server-overload is different from minimum bandwidth clustering problem and that our algorithm Earliest Maximal Clustering, also known as EMCL [17] is optimal in the static case. We perform extensive simulations over a wide range of parameters and a number of different clustering policies. We also demonstrate that the optimal re-computation period depends not only on the rates of arrival and interaction but also on the specific policy used.

Another contribution of this work is an attempt to formulate the dynamic stream clustering problem as a Stochastic Dynamic Programming problem, although that resulted in a few negative results. To the best of our knowledge, this is the first approach of its kind in this domain, and we believe that our preliminary findings will spur the VoD research community to find better solutions in this space. We also present an approximate analytical technique for predicting the average number of streams in such an aggregation system in steady state, and demonstrate that the analytical results are reasonably close to the simulation results.

The remainder of the chapter is organized as follows. In Section 2, we formulate the clustering problem mathematically, and consider optimal algorithmic solutions. Section 3 introduces a stochastic dynamic programming approach to tackle the *dynamic* stream clustering problem. In Section 4 we present the results of simulations of heuristic and approximate algorithms for clustering. In section 5 we conclude with our recommendations of clustering algorithms that should be used for aggregation in interactive VoD systems.

2 Stream Clustering Problems

Here we characterize the space of on-demand video stream clustering problems by formulating a series of sub-problems. We provide optimal solutions for some of these sub-problems and present heuristic solutions derived from these optimal approaches for the remainder.

2.1 General Problem Description

A centralized video-server stores a set of k movies $M = \{M_1, M_2, \ldots M_k\}$, each having lengths $\ell_1, \ell_2, \ldots, \ell_k$ respectively. It accepts requests from a pool of subscribers for a subset of these stored movies and disseminates the content to the customers over a delivery network which supports group communication (e.g., multicast). Assume that the movie popularity distribution is skewed (non-uniform) obeying a probability distribution $P(.)$. Each *popular* movie can be played at two different content progression rates r (normal speed) and $r + \Delta r$ (accelerated speed). Accelerated versions of these movies are created by discarding some of the less important video frames (e.g., B frames in the MPEG data stream) by the server on the fly. If the frame dropping rate is maintained below a level of $\approx 7\%$, the minor degradation of quality is not readily perceivable by the viewer [16, 4].

There is an opportunity to reclaim streaming data bandwidth if skewed streams of identical movies are re-aligned and consolidated into a single stream. This can be achieved by the server attempting to change the rate of content progression (the playout rate) of one or both of the streams.

There are several means of delivering the same content to a group of users using a single channel. Multicast is a major vehicle for achieving this over the IP networks. But because it has not been widely deployed in the general Internet, another scheme called IP Simulcast can be used to achieve the same goal by relaying or repeating the streams along a multicast tree [9]. If the medium has broadcasting capabilities

(such as cable TV channels) multicast groups can be realized by means of restricted broadcast.

The term *clustering* refers to the action of creating a schedule for merging a set of skewed identical-content streams into one. The schedule consists of content progression directives for each stream at specific time instants in the future. For large sets of streams, the task of generating this schedule is computationally complex and is exacerbated by the dynamics of user interactions to achieve VCR functions such as stop, rewind, fast-forward, and quit. It is clear that efficient algorithms for performing this clustering are needed for large sets of streams. The term *merging* refers to the actual process of following the clustering algorithm to achieve the goal.

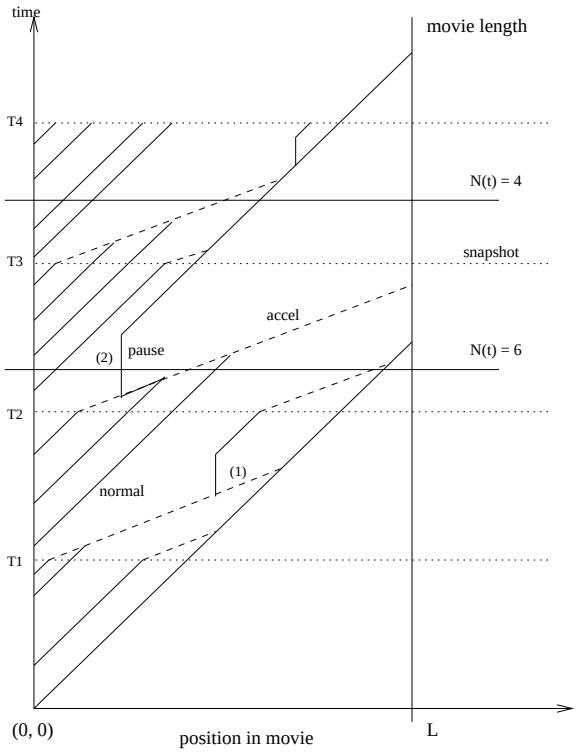


Figure 1: Aggregation of Streams

A general scenario for aggregation of video streams with clustering is outlined below. At time instants T_i the server captures a snapshot of the stream positions

– the relative skew between streams of identical content – and tries to calculate a clustering schedule using a clustering algorithm `CA` for aggregating the existing streams. Aggregation is achieved by accelerating a trailing stream towards a leading one and then merging whenever possible. Changing the rates of the streams is assumed to be of zero-cost and the merging of two streams is assumed to be seamless. Viewers, however, are free to interact via VCR functions are cause breakouts of the existing streams. That is, new streams usually need to be created to track the content streaming required for the VCR function. Sometimes these breakouts are clusters themselves when more than one viewer initiates an identical breakout. The interacting customers are allocated individual streams, if available, during their interaction phase but are attempted to be aggregated at the next snapshot epoch. We restrict our discussion to the VCR actions of `fast-forward, rewind, pause, quit`.

Figure 1 illustrates the merging process. Here the lifespans of several streams receiving the same content are plotted over time. The horizontal axis denotes the position of a stream in the program and the vertical axis is the time line. The length of the movie is assumed to be L as indicated by the vertical line on the right. Streams are shown in one of the following three states: normal playback, accelerated or paused. Accelerated streams have a steeper slope than the normal streams because they move faster than the latter. Paused streams have infinite slope because they do not change their position with time. Between time 0 and T_1 all arriving streams progress at a normal rate.

The first snapshot capturing positions of existing streams occurs at time T_1. At this time a clustering algorithm is used to identify the clusters of streams that should be merged into a single stream (per cluster). The merging schedule is also computed within each cluster. Different algorithms can be applied for performing the clustering and merging (see Section 4). When new users arrive in between snapshots they are advanced at a normal rate until the next snapshot when the merging schedule is computed again and the streams are advanced according to the new clustering schedule. The clusters and the merging schedule are computed under an assumption

that no interactions will occur before all the streams merge. But because the system is interactive, users can interact in the middle. If a user interacts, on a stream which is carrying multiple users, a new stream is allocated while the others on the stream continue. Case (1) in the figure demonstrates this phenomenon. However, if the only user on a stream interacts, the server stops sending data on the previous stream because it does not carry any user anymore. This is illustrated by case (2) in the figure. $N(t)$ denotes the number of streams existing in the system at a time instant t.

A state of a non-interacting stream S_i can be represented by a position-rate tuple (p_i, r_i), where $p_i \in \Re^+$ is the program position of the i^{th} stream, measured in seconds from the beginning of the movie and $r_i \in \{r, r + \Delta r\}$ is the content progression rate. The state equation is given by:

$$p_i(t + \Delta t) = p_i(t) + r_i(t)\Delta t \tag{1}$$

and $r_i(t)$ is determined by the clustering algorithm `CA`. Note that in this discussion, time is measured in continuous units for simplicity. In reality, time is discrete as the smallest unit of time is that required for the play-out of one video frame (approximately 33 ms for NTSC-encoded video).

As an example of a merge, consider streams S_i (rate r) and S_j (rate $r + \Delta r$). If they can merge in time t, then the following will hold:

$$p_i + rt = p_j + (r + \Delta r)t \tag{2}$$

$$t = \frac{p_i - p_j}{\Delta r} \tag{3}$$

The most general problem is to find an algorithm `CA` that minimizes $\frac{1}{T}\int_{\tau}^{\tau+T} N(t)dt$; the average number of streams (aggregated or non-aggregated) in the system in a time interval $[\tau, \tau + T]$ for given arrival and user interaction rates. The space of optimization problems in this aggregation context is explored next.

2.2 Clustering Under a Time Constraint

There is a streaming bandwidth overload scenario that exists when too many users need unique streams from a video server. This scenario arises when many users interact simultaneously, but distinctly, or when a portion of a video server cluster fails or is taken offline. The objective of a solution for this scenario is to recover the maximum number of channels possible within a finite time budget, by clustering outstanding streams. We assume that interactions are not honored during the recovery period because the system is already under overload.

Maximizing the number of channels recovered is desirable because it enables us to restore service to more customers and possibly to admit new requests. Recovery within a short period of time is necessary since customers will not wait indefinitely on failure. It is possible to mask failures for a short period of time by delivering secondary content such as advertisements [17]. This period can be exploited to reconfigure additional resources. However reconfigurability adds substantially to system costs and additional resources may not be available.

Optimal stream clustering under failure can be solved easily in polynomial time and a linear-time algorithm using either content insertion or rate adaptive merging is described in reference [17]. This algorithm, called EMCL (Earliest Maximal CLuster) starts from the leading stream and clusters together all streams within the time budget, then repeats the process by starting from the next stream ahead until the the final, trailing stream, is reached. A more general class of one-dimensional clustering problems has been shown to be polynomially solvable [12].

Three points are worth noting here. Firstly, the EMCL algorithm can be applied when both content insertion and rate adaptation are used. The number of channels recovered by the algorithm is non-decreasing with increase in the time budget. Using both techniques has the same effect as increasing the time budget when using one technique alone. Therefore we can apply the same algorithm. We will see later that this property does not generalize to other optimization criteria.

Secondly, this algorithm is not affected if some leading streams will reach the end of the movie before clustering. By accelerating all streams that are within the time budget from the end of the program and then computing the clustering for the remaining streams, we can achieve an optimal clustering.

Thirdly, iterative application of this algorithm does not provide further gains in the absence of interactions, exits, or new arrivals. In the dynamic case, the average bandwidth, in number of channels, used during clustering is of consequence. We consider this average bandwidth minimizing constraint and dynamicity in the following sections.

2.3 Clustering to Minimize Bandwidth

Another goal for clustering is to minimize the average bandwidth required in the provisioning of a system. This can be measured by the number of channels per subscriber and is more interesting and applicable from a service aggregation perspective. A special case of this problem occurs when we wish to construct a merging schedule at some instant that uses minimum bandwidth to merge all streams into one to carry a group of users. Ignoring arrivals, exits, and break-aways due to interaction, the problem has been approached heuristically by Lau et al. [18] and analytically by [3].

The problem can be readily transformed into a special case of the Rectilinear Steiner Minimal Arborescence (RSMA) problem [3], which is defined by Rao et al., [21] as:

> Given a set N of n nodes lying in the first quadrant of E^2, find a minimum length directed tree (called Rectilinear Steiner Minimal Arborescence, or RSMA) rooted at the origin and containing all nodes in N, composed solely of horizontal and vertical arcs oriented from left to right and from bottom to top.

The transformation is illustrated in Figure 2 with a minor difference that an *up*

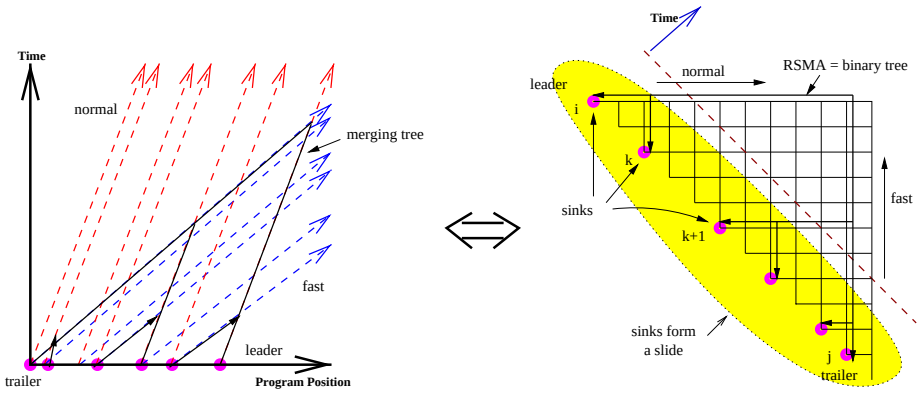


Figure 2: Minimum Bandwidth Clustering

arborescence – a directed tree with arcs directed towards the root – is used instead. In the figure on the left, the shaded dots denote the stream positions at the time instant corresponding to the beginning of the snapshot. Each stream has a choice of either progressing at a normal speed (larger slope in the figure) or at an accelerated speed (smaller slope). In the optimal tree, the absolute trailer must always be accelerated and the absolute leader must always be retarded, and the two lines corresponding to them will meet at the root. In Steiner tree terminology the shaded dots ($\in N$) are called *sinks*. The RSMA is denoted in the figure on the right as a rooted directed tree (the arrow directions have been reversed for a better intuitive feel).

In the RSMA formulation, *normal* streams move horizontally whereas the *accelerated* streams move vertically. The leader always remains horizontal and the trailer vertical. The sinks lie on a straight line of slope = -1 and the final merge point which is the root of the RSMA is at the origin. This transformation leads to a special case of the RSMA problem in which the sinks form a *slide*. The slide is a configuration without a directed path from one sink to another (i.e., there are no sinks in the interior of the rectilinear grid (called a *Hanan grid*). Note that this special case of the problem has an *optimal sub-structure* that is lacking in the general case, and finding the min-cost RSMA spanning n sinks is equivalent to finding an optimal cost binary tree on n leaf-nodes or an optimal bracketing sequence. The number of

binary trees on n leaf-nodes (or RSMAs with n sinks forming a slide) is given by the $(n-1)^{st}$ Catalan number, $\frac{1}{n}\binom{2n-2}{n-1}$. Although this constitutes an exponential search space, the special structure of the problem allows us to compute an optimal binary tree using a dynamic programming (DP) algorithm of complexity $\mathcal{O}(n^3)$ [21]. One can easily verify that the RSMA for the particular placement of sinks in Figure 2 is a minimum length binary tree. We call the DP algorithm `RSMA-slide` in this chapter.

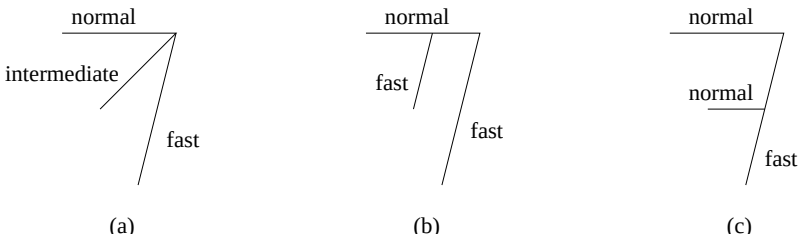


Figure 3: Merging with More than Two Rates

One point to be noted here is that from an optimality standpoint, more than two content progression rates are not needed. Fig. 3 illustrates the situation. A line segment with a higher Cartesian slope has a higher content progression rate. In Fig. 3(a), three rates are used for content progression and the streams yield a single meeting point. Clearly the merging trees in Figs. 3(b-c) have lower cost in terms of bandwidth used than the previous one, because a merge occurs earlier in those two cases. Therefore, two content progression rates are sufficient for rate adaptation.

Now, we describe the RSMA-slide algorithm which is based on optimal binary trees [2]. Let L be the length of the movie that is served to streams $i, \ldots j$, with $p_i > p_j$. $P(i,j)$ denotes the optimal program position where streams i and j would merge and is given by:

$$P(i,j) = p_i, \quad i = j \tag{4}$$

$$= p_i + r \times \frac{p_i - p_j}{\Delta r}, \quad i \neq j \tag{5}$$

Let $T(i,j)$ denote an optimal binary tree for merging streams $i, \ldots, j$. Let $Cost(i,j)$ denotes the cost of this tree. Because this is a binary tree, there exists a point k

such that the right subtree contains the nodes $i, \ldots, k$ and the left subtree contains the nodes $k+1, \ldots, j$. From the principle of optimality, if $T(i,j)$ is optimal (has minimum cost) then both the left and right subtrees must be optimal. That is, the right and left subtrees of $T(i,j)$ must be $T(i,k)$ and $T(k+1,j)$. The cost of this tree is given by

$$\begin{aligned} Cost(i,j) &= L - p_i, \quad i = j & (6) \\ &= Cost(i,k) + Cost(k+1,j) - \max(L - P(i,j), 0), \quad i \neq j & (7) \end{aligned}$$

and the optimal policy merges $T(i,k^*)$ and $T(k^*+1,j)$ into $T(i,j)$, where k^* is given by

$$k^* = \text{argmin}_{i \leq k < j}\{Cost(i,k) + Cost(k+1,j) - \max(L - P(i,j), 0))\} \quad (8)$$

Here $Cost(i,k)$ and $Cost(k+1,j)$ are the costs of the right and the left subtrees respectively, calculated until the end of the movie.

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

Algorithm RSMA-slide
{
  for ($i = 1$ to n)
    initialize $P(i,i)$, $Cost(i,i)$ and $T(i,i)$ from equations 4 and 6

  for ($p = 1$ to $n-1$)
    for ($q = 1$ to $n-p$)
      Compute $P(q,q+p)$, $Cost(q,q+p)$ and $T(q,q+p)$ from equations 5, 7 and 8
}

There are $\mathcal{O}(n)$ iterations of the outer loop and $\mathcal{O}(n)$ iterations of the inner loop. Additionally, determination of $Cost(i,j)$ requires $\mathcal{O}(n)$ comparisons in the *argmin* step. Hence, the algorithm RSMA_Slide has a complexity of $\mathcal{O}(n^3)$. Therefore, we

conclude that the static clustering problem has an optimal polynomial time solution, in contrast to the conclusion of Lau et al. [18]. Note that in practice, n is not likely to be very high, thus making the complexity acceptable.

Although optimal binary trees are a direct way to model this particular problem (where there are no interactions), the RSMA formulation offers more generality as we shall explain in later sections. Recently, Shi et al. proved that the min-cost RSMA problem is NP-Complete [23]. However, before RSMA was proved to be NP-Complete, Rao et al. had proposed a 2-approximation algorithm of time complexity $\mathcal{O}(n \log n)$ for solving the RSMA [21].

Minimum bandwidth clustering can be used as a sub-scheme to clustering under a time budget, so that channels are released quickly thereby accommodating some interactivity. This technique can speed up computations in a dynamic scenario, in which cluster re-computation is performed on periodic snapshots. The time-constrained clustering algorithm has linear complexity and the dynamic programming algorithm of cubic complexity needs to be run on a smaller number of streams. Simulation results for these are shown in Section 4.

2.4 Clustering with Arrivals

Relaxing the preceding problem to include stream arrivals, but not exits nor other interactions, we consider continue the investigation of the problem space. Suppose we could have a perfect prediction of new stream arrivals. The problem again transforms to the RSMA-slide formulation. Although the sinks do not lie on a straight line, they would still form a slide (e.g., the two points at the bottom right corner of Fig. 4 form a slide along with the points lying towards their left in the exterior of the grid). Therefore, a variant of the RSMA-slide algorithm of Section 2.3 can be used here. The cost calculation can differ in this case because two points can merge by following paths of unequal length. For example, the two predicted points in Fig. 4(b) are merged by line segments of different lengths. After the RSMA is calculated,

a corresponding merging tree can be found for the original formulation as shown in Fig. 4(a).

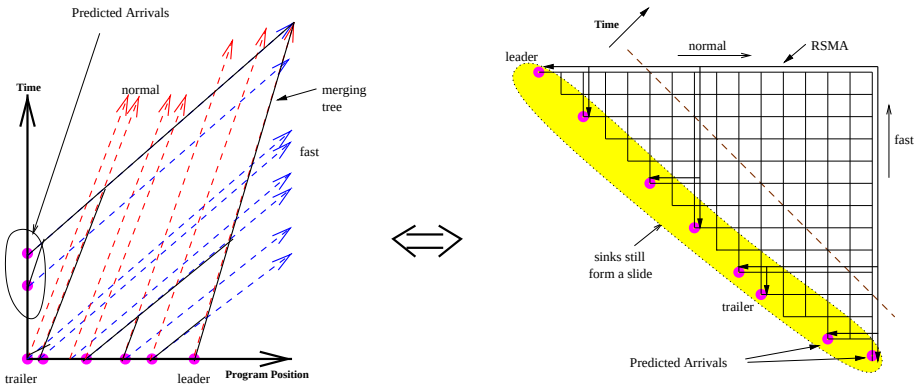


Figure 4: Clustering with Predicted Arrivals

Alternatively, we can batch streams for a period equal to or greater than the maximum window for merging. In other words, the users that have newly arrived can be made to wait for an interval of time for which the previously arrived streams are being merged. Although this can reduce the problem to the preceding one, in practice, this can lead to loss of revenue from customer reneging due to excessive waiting times.

Aggarwal et al. [2] instead use a periodic re-computation approach with an initial greedy merging strategy. They show that if the arrival rate is fairly constant, a good re-computation interval can be determined. If a perfect predictor were available (e.g., via advance reservations), then better performance can be achieved by using the exact RSMA-slide algorithm.

2.5 Clustering with Interactions and Exits

Here we show how a highly restricted version of the problem with certain unrealistic assumptions is transformable to the general RSMA problem. We assume a jump-type interaction model (i.e., a user knows where to fast forward or rewind to, and resumes

instantaneously). We also assume that the leading stream does not exit and that interactions do not occur in singleton streams (which are alone in their clusters). Furthermore we assume that a perfect prediction oracle exists which predicts the exact jump interactions. Our "perfect predictor" identifies the time and the program position of all user interactions and their points of resumption.

Fig. 5 shows a transformation to the RSMA problem in the general case. The shaded points in the interior of the grid depict predicted interactions. The problem here is to find a schedule that merges all streams taking into account all the predicted interactions and consumes minimum bandwidth during the process. This problem is isomorphic to the RSMA problem, which has recently been shown to be NP-Complete [23]. Also, the illustration shows that the RSMA is no longer a binary tree in this case.

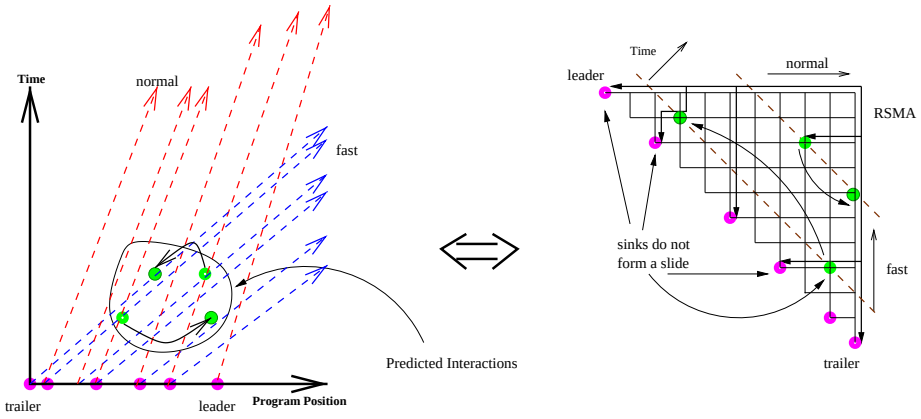


Figure 5: Clustering Under Perfect Prediction

Stream exits can occur due to the ending of a movie or a user quitting from a channel which has only one associated user. Exits and interactions by a user, alone on a stream, result in truncation and deletion of streams respectively, and cause the RSMA tree to be disconnected and form a forest of trees. The above model is not powerful enough to capture these scenarios. Perhaps a generalized RSMA forest can be used to model these situations. However, the RSMA forest problem is expected to be at least as hard as the RSMA problem.

An approximation algorithm for RSMA which performs within twice the optimal is available using a "plane sweep" technique. But in a practical situation in which a perfect prediction oracle does not exist, knowing the arrivals, and especially interactions, beforehand is not feasible. Therefore the practical solution is to periodically capture snapshots of the system and base clustering on these snapshots. The source of sub-optimality here is due to the possibility that a stream could keep on following its path as specified by the algorithm at the previous snapshot even though a stream with which it was supposed to merge has interacted and does not exist nearby anymore. Only at the next snapshot does the situation get clarified as the merging schedule is computed again. Also note that causal event-driven cluster recomputation will perform worse than the perfect oracle-based solution. For any stochastic solution to be optimal and computationally tractable, the RSMA must be solvable by a low order polynomial time algorithm. The important point here is that because we do not have a low cost predictive solution in interactive situations (the complexity of RSMA is unknown), we must recompute the merging schedules periodically to remain close to the optimal.

Note that an algorithm with time complexity $\mathcal{O}(n^3)$ (for RSMA-slide) *may be* expensive for an on-line algorithm and it is worthwhile comparing it with its approximation algorithm which runs faster, producing worst-case results with cost within twice the optimal. In the simulations section (Section 4), we compare both the algorithms under dynamic situations. Rao et al. [21] give a polynomial-time approximation algorithm based on a plane-sweep technique which computes an RSMA with cost within twice the optimal. The algorithm is outlined below.

If the problem is set in the first quadrant of E^2 and the root of the RSA is at the origin, then for points p and q belonging to the set of points on the rectilinear grid of horizontal and vertical lines passing through all the sinks, the parent node of points p and q is defined as $parent(p, q) = (\min(p.x, q.x), \min(p.y, q.y))$, and the rectilinear norm of a point $p = (x, y)$ is defined by $\|(x, y)\| = x + y$.

Algorithm RSMA-heuristic($\mathcal{S}$:*sorted stream list*)
{
  Place the points in $\mathcal{S}$ on the RSA grid
  `while` ($\mathcal{S} \neq \phi$)
    Find 2 nodes $p, q \in \mathcal{S}$ such that $\|parent(p,q)\|$ is maximum
    $\mathcal{S} \leftarrow \mathcal{S} - \{p,q\} \cup parent(p,q)$
    Append the branches $parent(p,q) \rightarrow p$ and $parent(p,q) \rightarrow q$ to the RSA tree.
  `end`
}

The algorithms yields an heuristic tree upon termination. Because it is possible to store the points in a *heap* data structure, we can find the parent of p and q with the maximum norm in $\mathcal{O}(\log n)$ time and hence the total time complexity is $\mathcal{O}(n \log n)$.

The fact that the static framework breaks down in the dynamic scenario is not surprising. Wegner [27] suggests that interaction is more powerful than algorithms and such scenarios may be difficult to model using an algorithmic framework.

2.6 Clustering with Harder Constraints and Heuristics

In practice, stream clustering in VoD would entail additional constraints. We list a few of them here:

- Limits on total duration of rate adaptation per customer
- Critical program sections during which rate adaptation is prohibited
- Limited frequency of rate changes per customer
- Continuous rather than "jump" interactions
- Multiple classes of customers with varying pricing policies

The inherent complexity of computing the optimal clustering under dynamicity, made even harder by these additional constraints, warrants search for efficient heuristic

and approximation approaches. We propose a number of heuristic approaches which are worthy of experimental evaluation. We evaluate these approaches experimentally in Section 4.6.

- **Periodic** Periodically take a snapshot of the system and compute the clustering. The period is a fixed system parameter determined by experimentation. Different heuristics can be used for the clustering computation:
  - Recompute RSMA-slide (optimal binary tree) using $\mathcal{O}(n^3)$ dynamic programming algorithm.
  - Use the $\mathcal{O}(n \log n)$ RSMA heuristic algorithm.
  - Use the $\mathcal{O}(n)$ EMCL algorithm to release the maximum number of channels possible.
  - Follow EMCL algorithm by RSMA-slide, heuristic RSMA-slide or all merge with leader.
- **Event-triggered** Take a snapshot of the system whenever the number of new events (arrivals or interactions) exceeds a given threshold. In the extreme case, recomputation can be performed at the occurrence of every event.
- **Predictive** Predict arrivals and interactions based on observed behavior. Compute the RSMA based on predictions.

2.7 Dependence of Merging Gains on Recomputation Interval

In this section, we use analytical techniques to investigate the channel usage due to $RSMA$ in the steady state. The modeling approach is more completely described in reference [15].

First, we can approximate expression for the number of streams needed to support users watching the same movie in a periodic snapshot-RSMA based aggregation

system as a function of time. We consider the non-interactive case here. In a typical recomputation interval R, new streams arrive (one user per stream) at a rate λ_a; the older streams are captured in the previous snapshot captured at the beginning of the current recomputation interval are merged using RSMA. Now, to make the calculation tractable, we assume that the process of capturing snapshots enforces a "natural clustering" of streams in the sense that the streams being merged in the current recomputation interval are less likely to merge with streams in the previous recomputation interval, at least for large values of R. This assumption is reasonable because last stream of the current cluster (the trailer) accelerates while the leading stream of the previous cluster (leader) progresses normally, thus resulting in a widening gap between end-points of clusters.

We model one recomputation interval R, and then consider the effect of a sequence of R's by a summation. Again, we make a very simplistic assumption to make the problem tractable. We assume that the streams in a snapshot are equally spaced and the separation is $\frac{1}{\lambda_a}$. In this situation, the RSMA is equivalent to progressively merging streams in a pairwise fashion. If L is the length of the movie and $K = 15$ is a factor arising due to the speed-up, the number of streams as a function of time from the beginning of the snapshot is given by:

$$C(t) = \begin{cases} 0 & : \quad t < 0 \\ \lambda_a t & : \quad 0 \leq t < R \\ \frac{\lambda_a R}{2^{\lfloor \log(\frac{\lambda_a (t-R)}{K} + 1) \rfloor}} \approx \frac{KR}{t - R + \frac{K}{\lambda_a}} & : \quad R \leq t < R + L \\ 0 & : \quad t \geq R + L \end{cases} \tag{9}$$

At any time instant t, the total number of streams $N(t)$ is built from contributions of a sequence of recomputation intervals. It is given by:

$$N(t) = \sum_{n=0}^{\infty} C(t - nR) \tag{10}$$

Note that only some of the terms in the above summation have a non-zero

contribution to the total sum. We are interested in calculating the average number of streams in steady state, in other words, in the value of $\lim_{T\to\infty} \frac{1}{T} \int_{t_0}^{t_0+T} N(t)dt$. But, it can be shown that N(t) is periodic with period R, hence we calculate the value of $\frac{1}{R} \int_{t_0}^{t_0+R} N(t)dt$, where t_0 signifies a time instant after steady state has been reached.

As we indicated before, $N(t)$ is non-zero only for a sequence of recomputation intervals; hence by appropriately taking contributions from relevant intervals, we get an expression for the average number of streams in the system in steady state:

$$\tilde{N}(R) = \frac{1}{R} \int_{t_0}^{t_0+R} N(t)dt \quad \approx \quad \frac{1}{2}\lambda_a R + K \ln(\frac{\lambda_a L + \lambda_a R + K}{K}) \tag{11}$$

We observe that $\tilde{N}$ has a linear component in R and a logarithmic (sub-linear) component in R, thus making the overall effect linear in R, especially for large value of R. We shall see in Section 4.6 that this agrees in behavior with the simulation results, although there are inaccuracies in the exact values of $N(t)$ due to the approximate nature of the model developed here.

3 Stochastic Dynamic Programming Approach

In this section, we attempt to model the interactive VoD system using stochastic dynamic programming (DP). We model the arrival, departure and interactions in the systems as *events*, and as in a standard DP based approach, we apply an optimization "control" at the occurrence of each event. Let us consider the system at time $t_0 = 0$. Suppose there are N users $u_1, u_2, \ldots, u_N$ in the system (for simplicity of analysis we assume all users receive the same program) at that particular time instant, and their program positions are given by a sorted position vector $P = (p_1, p_2, \ldots, p_N)$, respectively, where $p_1 \geq p_2 \geq \ldots \geq p_N$. Because this is an aggregation based system, the number of streams in the system, n, is less than the number of users. Because the

user position vector P is sorted, the stream position vector can be easily constructed from P in linear time. Therefore, the state of the system can be denoted in two ways: (1) by the user position vector P as described in the previous paragraph, or, (2) by a stream position vector $S = (s_1, s_2, \ldots, s_n)$ with ID's of users that each stream is carrying. We choose the latter representation for our analysis in this section.

In the following analysis, we assume that the N^{th} user arrived at time, t_0. Hence, $p_N = 0, s_n = 0$, and we must apply an optimization control at this time instant. An optimization control is nothing but a merging schedule which results in the minimization of the total number of streams in the long run under steady state. A merging schedule is a binary sequence $M = (r_0, r_1, \ldots, r_n)$ of length n, where r_i denotes the content progression rate of stream s_i, and $r_i \in \{r, r + \Delta r\}, \forall_i$.

We describe a model for the events in the system. Suppose, the next event occurs at time instant $t_0 + \tau$. The streams in the system progress according to the merging schedule M, which was prescribed at time t_0. The new position vector S' will be a function of the old position vector S, the merging schedule M, the inter-event time τ, and also the type of event. The size of the position vector S, increases, decreases or stays the same depending on the type of the event:

- **Arrival:** size increases by 1
- **Departure due to a user initiated quit:** size decreases by 1 (if the quit is initiated in a single-user stream) or stays the same (if the quit is initiated in a multi-user stream)
- **Exit due to the end of the program:** size reduces by 1, and number of users on the exiting stream.
- **Interaction:** size increases by 1 (if initiated in a multi-user stream) or stays the same (if initiated in a single-user stream)

3.1 Cost Formulation for Stochastic Dynamic Programming

We begin with a brief introduction to the stochastic DP approach [5] that we decided to utilize to model the system. The quantity that we want to minimize in the system is the number of streams, n in steady state. Using the DP terminology, we refer to this quantity as average *cost* incurred per unit time. The *total* average cost incurred can be represented by a sum of *short-term* cost and *long-term* cost per unit time. The former reflects the immediate gains achieved by the merging schedule M (i.e., before the next event occurs), whereas the latter reflects the long term gains due to M. If the current state (basically the position vector S at time t_0) is represented by i and the long-run cost incurred at state i is denoted by a function $h(i)$, then the total average cost per unit time at the current time instant t_0 is given by the Bellman's DP equation:

$$J^* + h(i) = \min_{u \in U(i)} \left[E[g(i,u)] + \sum_{j=1}^{t} p_{ij}(u) h(j) \right], \tag{12}$$

where, $U(i)$ is the set of possible controls (merging schedules, in our setting) that can be applied in state i, $g(i,u)$ is the short-term cost incurred due to the application of control u in state i, $E[g]$ denotes the expected value of the short-term cost, $p_{ij}(u)$ denotes the transition probabilities from state i to state j under control u and $h(j)$ is the long-run average cost at state j. t is a state that can be reached from all initial states and under all policies with a positive probability, and J^* is the optimal average cost per unit time starting from state t.

The Curse of Dimensionality Under this formulation, we quickly run into some problems. First, the state space is very large. If the maximum number of users in the system is N and the program contains L distinct program positions the number of states is clearly exponential in N, no matter how coarsely the program position are represented (obviously, this depends on the coarseness of representation). For fine grain state representations, the size of the state space grows exponentially as well. Also, the space of controls is exponential in n, where n denotes the number of streams

in the system, because each stream has two choices for the content progression rate. Furthermore, the long-run cost function $h(\cdot)$ is very hard to characterize for all states especially in the absence of a formula based technique. Currently we believe that $h(\cdot)$ can be characterized by running a simulation for long enough that states start to recur and then the long run average cost values can be updated for every state over a period of time. However, this technique is extremely cumbersome and impractical owing to the hugeness of the state space in a VoD system with a large dimension (N). Therefore, due to this *curse of dimensionality*, we believe that the DP method is impractical in its actual form and attempted to use it in a partial manner, which we explain in detail next.

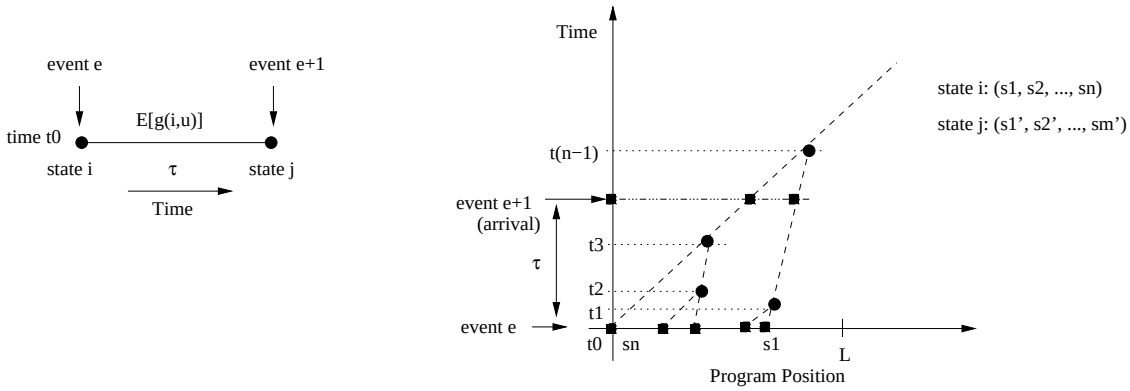


Figure 6: Event Model and Controls

Expected Cost Calculation Figure 6 demonstrates the basic model of events and controls. The left side depicts the occurrence of events e and $e+1$ for a section of the time-line. The inter-event time is τ and the system moves from state i to state j under control u and incurs a short-term expected cost denoted by $E[g(i,u)]$ which depends on the distribution of the random variable τ. We illustrate a technique for the calculation of this quantity below.

The control space is chosen carefully to prevent an exponential search and to yield an optimal substructure within a control. u is considered to be a feasible control if it represents a binary tree with $s_1, s_2, \cdots, s_n$ as leaves. Hence u is a function of

the state i. Although the control space $U(i)$ grows combinatorially with n, the use of the principle of optimality (as demonstrated later) results in a polynomial time algorithm for choosing the optimal control. On the right side of Figure 6, a control is shown as a binary tree which is a function of state i. We depict the intermediate merge points in the binary tree by $t_k : 1 \leq k \leq n-1; t_1 \leq t_2 \cdots \leq t_{n-1}$. $t_n = \frac{L-s_1}{r+\Delta}$ denotes the time that the leading stream will run. Here we assume that the leading stream goes fast after the final merge and that all n streams merge before the movie ends (i.e., $t_{n-1} < \frac{L-s_1}{r+\Delta}$). t_k's can be determined easily from the stream position vector (essentially state i) and the control under consideration, u.

We formulate the short-term cost function for a given state i and a control u as the cost per unit time between events e and $e+1$. For simplicity, we assume that $t_0 = 0$; because this contributes to an additive term in the expected cost formulation and it does not affect the comparison of costs under two different controls.

$$g(i,u) = \begin{cases} n & : \quad 0 = t_0 < \tau < t_1 \\ \frac{1}{\tau}(nt_1 + (n-1)(\tau - t_1)) & : \quad t_1 \leq \tau < t_2 \\ \vdots & : \quad \vdots \\ \frac{1}{\tau}(nt_1 + (n-1)(t_2 - t_1) + \cdots + (n-k)(\tau - t_k)) & : \quad t_k \leq \tau < t_{k+1} \\ \vdots & : \quad \vdots \\ \frac{1}{\tau}(nt_1 + (n-1)(t_2 - t_1) + \cdots + 2(t_{n-1} - t_{n-2}) + (\tau - t_{n-1})) & : \quad t_{n-1} \leq \tau < t_n = \frac{L-s_1}{r+\Delta} \end{cases} \tag{13}$$

The general (k^{th}) branch in the above cost formulation simplifies to the following:

$$\begin{aligned} g_k(i,u) &= \frac{1}{\tau}(nt_1 + (n-1)(t_2 - t_1) + \cdots + (n-k)(\tau - t_k)) \\ &= \frac{1}{\tau}\sum_{l=0}^{k} t_l + (n-k), \qquad t_k \leq \tau < t_{k+1} \end{aligned} \tag{14}$$

The only random variable in the above analysis is the inter-event time τ which is exponentially distributed because the occurrence of various events can be modeled by the Poisson distribution. In particular, if the arrivals occur with a mean rate λ_a, quits occur with a mean rate λ_q, and interactions occur with a mean rate λ_i,

and each is modeled by the Poisson distribution, then the aggregate event process is also Poisson with a rate $\lambda = \lambda_a + \lambda_q + \lambda_i$ (i.e., the mean inter-event times (τ) are exponentially distributed with mean λ). The expected cost under control u can be calculated in the following manner.

$$\begin{aligned} E[g(i,u)] &= \int_0^\infty g(i,u) f(\tau) d\tau = \int_0^\infty \lambda e^{-\lambda\tau} g(i,u) d\tau = \int_{t_0}^{t_n} \lambda e^{-\lambda\tau} g(i,u) d\tau \\ &= \sum_{k=0}^{n-1} \big[\int_{t_k}^{t_{k+1}} \lambda e^{-\lambda\tau} g_k(i,u) d\tau \big] = \sum_{k=0}^{n-1} E[g_k(i,u)], \end{aligned} \tag{15}$$

where $E[g_k(i,u)]$ is given by

$$\begin{aligned} E[g_k(i,u)] &= \int_{t_k}^{t_{k+1}} \lambda e^{-\lambda\tau} \big[\frac{1}{\tau} \sum_{l=0}^{k} t_l + (n-k) \big] d\tau = (n-k) \int_{t_k}^{t_{k+1}} \lambda e^{-\lambda\tau} d\tau + (\sum_{l=0}^{k} t_l) \int_{t_k}^{t_{k+1}} \frac{1}{\tau} \lambda e^{-\lambda\tau} d\tau \\ &= (n-k)[e^{-\lambda t_k} - e^{-\lambda t_{k+1}}] + \lambda (\sum_{l=0}^{k} t_l)[Ei(\lambda t_k) - Ei(\lambda t_{k+1})], \end{aligned} \tag{16}$$

where $Ei(t) = \int_t^\infty \frac{e^{-x}}{x} dx$. From Equation 15, after algebraic simplification, we yield a compact expression for the expected cost:

$$E[g(i,u)] = n - \sum_{k=1}^{n} e^{-\lambda t_k} + \lambda \sum_{k=1}^{n} t_k Ei(\lambda t_k) \tag{17}$$

Stochastic RSMA Having derived an expression for the expected short-term cost, we attempted to use it for choosing the optimal control. In other words, given that we are in state i, we find the control $u(i)$ which minimizes $E[g(i,u(i))]$. Instead of direct constrained minimization, we use the principle of optimality and the separability of terms in Equation 17 to represent the cost recursively as follows.

$$
\begin{aligned}
E[g(i,u)] &= 1, \quad n = 1 \\
&= E[g(i_L,u_L)] + E[g(i_R,u_R)] - (e^{-\lambda t_{n-1}} - e^{-\lambda t_n}) + \lambda(t_{n-1}Ei(\lambda t_{n-1}) - t_n Ei(\lambda t_n)), n > 1
\end{aligned} \tag{18}
$$

where u_L and u_R represent the left and the right subtrees of u, respectively, and i_L and i_R represent the corresponding portions of the state i. Because t_{n-1} is fixed for a given state i, a static DP approach that is very similar to the RSMA-slide approach described in Section 2 can be applied to produce the optimal u. The time complexity of this algorithm, called Stochastic RSMA, is $\mathcal{O}(n^3)$. Simulation results for the algoithm are shown in Section 4.6.

4 Simulations

In this section, we show simulations of the clustering algorithms for rate adaptive merging described earlier, and compare them with some heuristic algorithms which will be described later in this section.

4.1 Objective

Some of the algorithms described in Section 2 are optimal in the static case (i.e., in the absence of user interactions). In addition, no optimal algorithms are known for the dynamic scenario in which users arrive into the VoD system, interact, and depart. However, the behavior of such a system with the application of the aforementioned clustering and merging techniques can be studied by simulations. In these simulations, we investigate the gains offered by an ensemble of techniques under varying conditions. We attempt to answer the following questions:

- Which policy is best in the non-interactive scenario?

- Which policy is best under dynamic conditions in the long-run?
- What is effect of high arrival and interaction rates on these algorithms?
- What algorithm should be used when facing server overload?

4.2 The Simulation Setup

We treat the simulation as a discrete-time control problem. As outlined in Section 2, there are various ways in which we can control the system including: apply control with a fixed frequency, apply control with a fixed event frequency, or adapt control frequency depending on the event frequency. The latter being the hardest to implement. In this work, we have considered only the first method of periodic application of control.

The simulation setup consists of two logical modules: a discrete event simulation driver and a clustering unit. The simulation driver maintains the state of all streams and generates events for user arrivals, departures and VCR actions (fast-forward, rewind, pause and quit) with inter-event times obeying an exponential probability distribution. Here we have not considered a bursty model of interactions. A merging window (also called the re-computation period) is an interval of time within which a clustering algorithm attempts to release channels. Once every re-computation period, the simulation driver conveys the status of each stream to the clustering unit and then queries it after every simulation tick to obtain each stream's status according to the specified clustering algorithm. It then advances each stream accordingly. For instance, if the clustering unit reports the status of a particular stream to be accelerating at a given instant of time, the simulation driver advances that stream at a slightly faster rate of $\frac{16}{15}$ times that of a normal-speed stream.

The clustering unit obtains the status of all running streams from the simulation driver and computes and stores the complete merging tree for every cluster in its internal data structures until it is asked to recompute the clusters. On this basis,

Table 1: Simulation Parameters

Parameter	Meaning	Value
M	Number of movies[3]	100
L	Length of a movie	30 min
W	Re-computation window	$10 - 1000$ sec
λ_{arr}	Mean arrival rate[4]	$0.1, 0.3, 0.7, 1.0\,\text{sec}^{-1}$
λ_{int}	Mean interaction rate[4,5]	$0, 0.01, 0.1\,\text{sec}^{-1}$
λ_{dur}	Mean interaction time[4]	5 sec

it supports querying on the status of any running non-interacting stream by the simulation driver after every simulation time unit.

4.3 Assumptions and Variables

The main parameters in our simulations are listed in Table 1. We simulated three types of interactions: *fast forward, rewind* and *pause*. λ_{int} is an individual parameter that denotes the rate of occurrence of each of the above interaction types (e.g., if $\lambda_{int} = 0.02\ sec^{-1}$, it means a fast-forward, or a rewind etc. occurs in the system once every 50 seconds). For simplicity we maintain λ_{int} the same for each type of interaction.

λ_{arr} and λ_{int} determine the number of users in the system at any given time. The user pool is assumed to be large. Also, we do not consider the probability of *blocking* of a requesting user here as we assume that the capacity of the system is greater than the number of occupied channels.

The content progression rate of a normal speed stream is set at 30 fps and that of an accelerated stream is 32 fps. The accelerated rate can be achieved in practice

[3]Movie popularities obey Zipf's law
[4]Exponential distribution assumed
[5]for each type of interaction

by dropping a B- picture from every MPEG group of pictures while maintaining a constant frame playout rate. Fast-forward and fast-rewind are fixed at $5X$ normal speed. Finally, the simulation granularity is 1 *sec* and the simulation duration is $8,000$ seconds.

We investigated the channel and bandwidth reclamation rates for each clustering algorithm and compared the results (Section 4.5).

4.4 Clustering Algorithms and Heuristics

We simulated six different aggregation policies and compared them:

EMCL-RSMA Clusters are formed by the EMCL algorithm and then merging is performed in each cluster by the RSMA-slide algorithm (Section 2).

EMCL-AFL Clusters are formed by the EMCL algorithm and then merging is performed by forcing all trailers in a cluster to chase their cluster leader.

EMCL-RSMA-heuristic Clusters are formed by the EMCL algorithm and then merging is performed by a RSMA-heuristic which is computationally less expensive than the RSMA-slide algorithm.

RSMA The RSMA-slide algorithm is periodically applied to the entire set of streams over the full length of the movie, but without performing EMCL for clustering.

RSMA-heuristic The RSMA-heuristic algorithm is periodically applied to the entire set of streams over the full length of the movie, but without performing EMCL for clustering.

Greedy-Merge-heuristic Merging is done in pairs starting from the top (leading stream) in the sorted list. If a trailer cannot catch up to its immediate leader within the specified window, the next stream is considered. At every merge, interaction, or arrival event, a check is performed to determine if the immediate

leader is a normal speed stream. If so, and merging is possible within a specified window, the trailing stream is accelerated towards the immediate leader. This process is continued throughout the window. GM is similar to Golubchik's heuristic [11] apart from the fact that it re-evaluates the situation at every interaction event as well.

The first three policies release the same number of channels at the end of an interval because they use the EMCL algorithm for creating mergeable clusters. But, the amount of bandwidth saved in the merging process is different in each case. EMCL-RSMA proves to be optimal over a given merging window, so it conserves the maximum bandwidth. EMCL-AFL is a heuristic and can perform poorly in many situations. EMCL-RSMA-heuristic, in contrast, uses an approximation algorithm for the RSMA-slide that guarantees a solution within twice the optimal [21]. Although our problem falls within the *slide* category and has an $\mathcal{O}(n^3)$ exact solution, we experiment with the approximation algorithm that runs in $\mathcal{O}(n \log n)$ time.

4.5 Results and Analysis

Here we discuss clustering gains (channel reclamation) and merging gains (bandwidth reclamation) due to the aforementioned algorithms. We classify the simulations into two categories:

4.5.1 Static Snapshot Case

The static snapshot case is applicable in overload situations. That is, when the server detects an overload, it wants to release as many channels as possible in a static merging window (one in which there are no user interactions). We vary the window size for a fixed number of users. Fig. 7(a) shows that the clustering gain increases as W increases. Also visible is that EMCL outperforms GreedyMerge over any static window of size W. In Fig. 7(b), compares the bandwidth gains due to the

Table 2: Arrival and Interaction Patterns

Arrivals / Interactions	NO	LOW	HIGH
LOW	$\lambda_{arr} = 0.1,\ \lambda_{int} = 0$	$\lambda_{arr} = 0.1, \lambda_{int} = 0.01$	$\lambda_{arr} = 0.1, \lambda_{int} = 0.1$
LOW-MEDIUM	$\lambda_{arr} = 0.3, \lambda_{int} = 0$	$\lambda_{arr} = 0.3, \lambda_{int} = 0.01$	$\lambda_{arr} = 0.3, \lambda_{int} = 0.1$
HIGH-MEDIUM	$\lambda_{arr} = 0.7, \lambda_{int} = 0$	$\lambda_{arr} = 0.7, \lambda_{int} = 0.01$	$\lambda_{arr} = 0.7, \lambda_{int} = 0.1$
HIGH	$\lambda_{arr} = 1.0, \lambda_{int} = 0$	$\lambda_{arr} = 1.0\lambda_{int} = 0.01$	$\lambda_{arr} = 1.0, \lambda_{int} = 0.1$

three merging algorithms. The merging gain increases with increase in W. Also, over any static window in the graph, the gains due to EMCL-RSMA and EMCL-RSMA-heur are almost identical! Although EMCL-RSMA is provenly optimal over a static window, the RSMA heuristic does as well in most cases (the reason for this may be attributed to the parameters of the particular inter-arrival distribution). Although cases can be constructed in which the heuristic produces a result that is twice the optimal value, in most practical situations it performs as well as the exact algorithm at a lower computational cost.

Fig. 7(c) shows the number of users varied for a fixed window size of 100 s comparing GM and EMCL. Clearly, EMCL performs better than GM and their clustering gains remain almost constant throughout the curves although the gap between the two appears to increase as U increases. Fig. 7(d) shows that EMCL-RSMA and its heuristic version perform equally well and clearly reclaim more bandwidth than EMCL-AFL. Even in this case, the percentage gains appear to remain constant as U increases.

4.5.2 Dynamic Case

The dynamic case is the general scenario in which users come into the system, interact, and then leave. Here, we simulate over a time equivalent of approximately $4\frac{1}{2}$ lengths of the movies.

We study the behavior of cumulative channel reclamation gains due to the above algorithms as a function of time for cases with varying rates of user arrival and interactions. We considered 12 different arrival-interaction patterns as shown in Table 2.

For each of the above cases and for each of the six different aggregation policies, we ran the simulations for eight different re-computation window sizes(W): 10, 25, 50, 100, 200, 500, 750 and 1,000 seconds. Fig. 8 shows the steady-state behavior of the system for $\lambda_{arr} = 0.7$ and $\lambda_{int} = 0.1$. We can clearly see that RSMA outperforms all other policies by a large margin for small and medium values of W. For $W = 100$ *sec*, RSMA reclaims about 400 channels from $1,250$ channels after the steady state is reached thus yielding gains of about 33%. This essentially means that for the particular type of traffic ($\lambda_{arr} = 0.7$ and $\lambda_{int} = 0.1$), we do not need more than 850 channels to serve $1,250$ streams.

For smaller W, other policies provide diminished channel gains. But as the re-computation interval is increased, RSMA begins to suffer and EMCL based policies and GM begin to perform well. This is because RSMA takes a snapshot of the whole system at the beginning of a recomputation interval and advances streams according to the *RSMA tree* computed from that snapshot. In the situation in which the user arrival rate is moderately high and the interaction rate is high as well, for high values of W, a many streams come in and interact between two consecutive snapshots and thus the gains due to RSMA drop. In contrast, GM tries to reevaluate the situation at every arrival and interaction event, so it performs very well.

Our speculation and conclusion is that different policies will perform at their best for different values of W and indeed, we found this to be true. Fig. 9 shows how the channel gains vary with the value of the recomputation window. For consistency, we consider the case with $\lambda_{arr} = 0.7$ and $\lambda_{int} = 0.1$. The graphs for the other cases are similar in shape and have not been shown due to space limitations. In the steady state, the system has $1,250$ users on average. For small values of W (≤ 500 *secs*), EMCL-based schemes and GM do not perform well, whereas, RSMA and RSMA-heur

perform well. However, for larger values of W ($> 500\,secs$), EMCL-based schemes and GM perform well and RSMA-heur is highly sub-optimal. Although RSMA shows reduced gains for higher values of W, it is still superior to EMCL-based algorithms. GM outperforms all algorithms, including RSMA for $W \geq 750$ due to the reason mentioned in the previous paragraph. But GM is more CPU intensive as it reacts to every arrival, interaction, and merge event. In that sense, it is not a true snapshot algorithm like the others, and is not a scalable solution under heavy rates of arrival and interaction although it may perform better than the other algorithms in such situations.

Each policy attains its best results for different values of W. RSMA performs best for $W = 10\,sec$; GM performs best for $W = 1,000\,secs$ and the EMCL based algorithms perform best for $500 \leq W \leq 750$. Fig. 10 plots the best channel gain achieved by a policy for every class of traffic (see 2). In the figure, there are four sets of curves with three curves each. Each set corresponds to an arrival rate because λ_{arr} is the most dominant factor affecting the number of users in the system. The mean numbers of users in the system for $\lambda_{arr} = 0.1, 0.3, 0.7$ and 1.0 are $180, 540, 1,250$ and $1,800$ respectively. The mean clustering gains increase significantly with increase in the arrival rate.

For each of the four sets, the channel gains are almost identical for $\lambda_{int} = 0$ and $\lambda_{int} = 0.01$ but are less for $\lambda_{int} = 0.1$. That is not far from expectations as a high degree of interaction causes each algorithm to perform worse. More importantly, the RSMA algorithm emerges as the best overall policy. As λ_{arr} increases, the gap between RSMA and the other algorithms widens.

Also, there little difference in the gains due to EMCL-RSMA and its heuristic counterpart, although there is a vast difference between the gains due to RSMA and those due to the RSMA heuristic. This is because EMCL breaks a set of streams into smaller groups and the EMCL-RSMA heuristic performs almost identically well as EMCL-RSMA for smaller groups. But, in case of RSMA, the number of streams is large (EMCL has not been applied to the stream cluster), so the

sub-optimality of the heuristic appears. Note that the RSMA-slide algorithm of complexity $\mathcal{O}(n^3)$ is affordable in practical situations because it is invoked only once in every recomputation period. As a reference point, the running time of the algorithm on an Intel Pentium Pro 200 MHz PC running the Linux operating system is well below the time required to playout a video frame; therefore no frame jitter is expected on modern playback hardware.

¿From the above simulations, we conclude the following:

- EMCL-RSMA is the optimal policy in the static snapshot case as it reclaims the maximum number of channels with minimum bandwidth usage during merging. Thus it is the ideal policy for handling server overloads.
- In an i-VoD system showing 100 movies, for dynamic scenarios with moderate rates of arrival and interaction, periodically invoking RSMA with low re-computation periods yields the best results.
- In realistic VoD settings, the arrival rate plays a greater role than the interaction rate (except for interactive game settings where aggregation is not a feasible idea). High arrival rates result in more streams in the system thus creating better scope for aggregation.
- Higher degrees of interaction result in lesser clustering gains for every algorithm.
- With increasing rates of arrival, RSMA's edge over other algorithms increases.
- The EMCL-RSMA heuristic performs almost as well as EMCL-RSMA although the RSMA-heuristic performs very badly as compared to RSMA.

4.6 Comparisons Between RSMA Variants

Here we compare the performance of the various forms of the RSMA-slide algorithm, namely, periodic snapshot RSMA (`Slide`), event-triggered RSMA (Section 2.6),

stochastic RSMA (Section 3), and predictive RSMA (Section 2.6). We consider a single movie of length $L = 2$ hours.

First, we evaluate the performance of `Slide` in the "arrival-only" situation because it lends itself to easy comparison with the analytical framework described in Section 2.7. The behavior of the channel gains is depicted in Fig. 11(a) as a function of the recomputation interval R. The simulation results show that as R is increased, the channel gain degrades as a linear function of R. The analytically estimated channel gains as described in Section 2.7 are plotted on the same graph. From the graph we see that the analytical estimation comes reasonably close to the simulation results. Although the analytical technique underestimates the simulation results by approximately 10 streams, the slopes of the two curves are almost the same. The slope is very close to the analytical estimate of $\frac{1}{2}\lambda_a$ (Equation 11). The underestimation can be attributed primarily to the assumption that the recomputation results in formation of natural clusters, and the streams in two different clusters do not merge. This assumption may not be valid due to a possible effect of streams in one recomputation interval on those in the next or previous one.

Next, we consider the threshold based, event-triggered RSMA, in which recomputation is not done periodically but only when the number of events exceeds a threshold. Fig. 11(b) shows the variation of the gains with the increase in the event count threshold. The case in which the threshold is 0 corresponds to the pure event-triggered RSMA where recomputation happens at the occurrence of each event. As expected, the gains decrease as the threshold is increased.

In summary, Table 3 shows the number of users and streams for each variant of RSMA in steady state. The `Slide` refers to the `RSMA-slide` with $R = 10$ sec. The rationale for using this value is that the arrival rate considered is 0.1 /sec, and hence on average there will be an arrival every 10 seconds. By "perfect prediction," we intend the following: to evaluate whether gains are achieved with this technique that warrant obtaining the difficult prediction of future events. Note that the RSMA can be computed optimally under perfect prediction if the events comprise arrivals

only, as shown in Section 2.4. When the actual stream arrives later, it just follows the path computed by the predictive RSMA algorithm. Stochastic RSMA reacts to every arrival event and the RSMA tree is determined using the technique described in Section 3. The event-triggered RSMA algorithm corresponds to the case in which the event count threshold is 0.

Table 3: Comparison between Variants of RSMA ($\lambda_{arr} = 0.1, \lambda_{int} = 0, \lambda_q = 0$)

RSMA Variant	Avg. #Users	Avg. #Streams	Avg. #Users per Stream
RSMA Slide ($R = 10$ sec)	722	71	10.17
Event Triggered RSMA	722	71	10.17
Predictive RSMA	721	71	10.15
Stochastic RSMA	732	137	5.34
GreedyMerge	720	373	1.93

We see from Table 3 that with the exception of the stochastic RSMA, the other variants perform almost identically under the current set of conditions. Because the merging algorithm alters the effective service time of a user by manipulation of the content progression rates, it affects the number of users in the system directly. The random number generator in the simulator produced user arrivals with mean $\lambda_{effective} = 0.1033$/sec which is slightly higher than the intended mean of $\lambda_{arr} = 0.1$/sec. Hence the average number of users in steady state in a non-aggregated system should be $\lambda_{effective} \times L = 0.1033 \times 7200 = 744$, but due to aggregation, the effective service time for many streams is reduced when they progress in an accelerated fashion, hence the number of users in the system is reduced. Stochastic RSMA performs worse than the other RSMA variants because only the short-term cost is considered in the formulation. For comparison, we show the gains due to the deterministic "greedy" algorithm. It performs much worse than its stochastic greedy counterpart (i.e., stochastic RSMA) as well as the the other RSMA variants.

Among all RSMA variants, the `Slide` is the simplest to implement and is computationally less expensive than its event-triggered counterpart because it does not react to every

event. Predictive RSMA, in contrast, does not provide additional gains even with perfect prediction, hence the computational overhead involved in prediction is not justified. We conclude that the `Slide` algorithm yields the best results with the least computational overhead than its variants.

Fig. 12 shows the effect of varying degrees of interactivity on the RSMA slide algorithm as a function of R in steady state. We considered three different values of interaction rates: 0, 0.01 and 0.02 per second for each type of interaction respectively. The quit rate is held at a low constant value (0.001 per second) in order to prevent the number of users in the system from substantially varying. The linearity dependence on R is lost as the interaction rate increases – for $\lambda_{int} = 0$, the curve is practically linear, whereas for $\lambda_{int} = 0.02$, the curve is not linear. This can be attributed to random interactions such as fast-forward and rewind result in greater fluctuations in stream positions at snapshots, thus making the merging trees of adjacent snapshots less correlated. Non-linearities arise in all three curves for lower values of R because there is no natural clustering due to taking snapshots with low R, and hence the streams in adjacent temporal clusters can merge later. Our analytical estimation framework does not take such behavior into account. As R increases, the gap between the interactive case and the non-interactive case widens significantly. This phenomenon is not unexpected as a larger R means more interaction events in that interval, which in turn translates to the increase in the loss of gains that were achieved due to merging. Hence, as observed before, high degrees of interactivity can be handled by recomputing the RSMA frequently.

Table 4: Comparison between Variants of RSMA ($\lambda_{arr} = 0.1, \lambda_{int} = 0.01, \lambda_q = 0.001$)

RSMA Variant	Avg. #Users per Stream
RSMA Slide ($R = 6$ sec)	8.33
RSMA Slide ($R = 16$ sec)	8.23
Event Triggered RSMA	8.62
Stochastic RSMA	4.23

Table 4 shows the performance comparison of the different RSMA variants in the dynamic, interactive cases. In comparison with Table 3, the average number of users per stream is lower due to user interactions. The aggregate event rate is $\lambda_{arr} + \lambda_{int} + \lambda_q = 0.161$, and hence the mean inter-event time is $\frac{1}{0.161} \approx 6$ sec. Hence we use $R = 6$ in the RSMA-slide algorithm. The event-triggered RSMA performs slightly better than its slide counterpart, but not by a substantial margin. Expectedly, RSMA-slide with a slightly higher $R(16\,\text{sec})$ yields slightly worse results than RSMA-slide with $R = 6$. However, Stochastic RSMA performs much worse as compared to the other variants because it does not take the long-term cost into account. The event triggered version of RSMA starts performing better than the periodic version as interactivity increases, but it has higher computational overhead. Therefore, a system designer should evaluate the system parameters and evaluate the trade-offs between the desired level of performance and computational overhead, and choose a clustering algorithm accordingly.

5 Conclusion and Future Work

Server based aggregation schemes are important for scalable delivery of interactive VoD content to heterogeneous end-clients with varying memory and storage capacities. In this chapter, we described the modeling of *service aggregation by server-side stream clustering* in i-VoD systems. We investigated various optimality criteria arising in stream clustering in video-on-demand, and then identified an optimal algorithm (RSMA-slide) with time complexity $\mathcal{O}(n^3)$ for the "static stream clustering" problem. We also modeled a restricted version of the dynamic clustering problem (with user arrivals and interactions) by a general RSMA which has been recently proven to be NP-complete. Because the general RSMA formulation in the interactive case requires unreasonable assumptions and does not yield a polynomial time solution, we have modeled the dynamic problem using iterative invocation of the optimal algorithms applicable in the static case.

We demonstrated full-scale simulations of the dynamic scenario with users entering, interacting, and leaving the system. In the simulations, we performed six different aggregation policies and compared their clustering gains in the steady-state. We investigated each policy for several re-computation intervals (W), arrival and interaction rates (λ_{arr} and λ_{int} respectively), and observed that a particular policy performs at its best at different values of W, λ_{arr} and λ_{int}. Although EMCL_RSMA and its heuristic counterpart perform equally well in most situations, RSMA-slide outperforms its heuristic counterpart by a large margin because of the larger number of streams in a snapshot.

We also attempted to use stochastic dynamic programming to model the dynamic, interactive scenario, but ran into difficulties due to computational complexity. Although we were able to craft a closed-form representation of the short-term cost due to merging between two successive events, we were unable to characterize the long-term cost in this study. Unfortunately, optimization with short-term cost as the objective does not yield satisfactory results. However, we remain hopeful that further investigation in this direction can result in the discovery of a good long-term cost formulation.

We observed that periodic invocation of the RSMA-slide clustering algorithm with a small recomputation interval produces best results under moderately high rates of arrival and interactions. Its $\mathcal{O}(n^3)$ time-complexity is also justified because it is invoked only once in a recomputation period that is of the order of tens of seconds to a few minutes, and also because it yields much better gains than does its heuristic version. We analytically approximated channel usage due to RSMA-slide in non-interactive situations as a function of R, and then showed by simulation that its performance degrades linearly with increase in R; the analysis and simulation results were in reasonable agreement with each other.

Under high rates of arrival and interaction, we showed that RSMA with large re-computation periods yields highly suboptimal results, and heuristics that react to every event have superior performance. A greedy heuristic like GM may result in a

much larger number of undesirable rate changes in the system. We also evaluated the performance of other variants of the RSMA-slide algorithm and found that the event-triggered RSMA and RSMA-Slide with $R = \frac{1}{\lambda_a}$ perform almost equally well. The event triggered version performs well at the cost of a higher computational overhead. Hence we choose RSMA-slide with low to medium re-computation intervals to be the best overall clustering policy in moderately interactive situations. R can be tuned by the system designer to balance the trade-off between optimality and computational overhead.

There are a variety of unsolved problems related to the work described herein. Future work will consider the feasibility and complexity of distributed clustering schemes and the clustering of streams served from geographically displaced servers. If logical channels are not independent and share a common medium like IP multicast, new stream creation can impact existing streams. Specifically we have assumed QoS in-elasticity; interaction of these algorithms in the presence of heterogeneous and layered streams is interesting. Another important issue that remains to be resolved before clustering schemes can be deployed in practice is seamless splicing of video frames compounded by network delays and delays introduced by inter-frame coding as in MPEG. Moreover, blocking of users due to depletion of channels is not considered here; however, blocking and service denial is inevitable in a system with finite resources and its probability has to be minimized.

References

[1] K.C. Almeroth and M.H. Ammar, "On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service," *IEEE JSAC,* Vol. 14, No. 6, pp. 1110-1122, Aug. 1996.

[2] C.C. Aggarwal, J.L. Wolf and P.S. Yu, "On Optimal Piggyback Merging Policies for Video-on-Demand Systems," *Proc. ACM SIGMETRICS '96*, Philadelphia, PA, USA, pp. 200-209, May 1996.

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

[4] P. Basu, A. Narayanan, R. Krishnan and T.D.C. Little, "An Implementation of Dynamic Service Aggregation for Interactive Video Delivery," *Proc. SPIE MMCN '98*, San Jose, CA, USA, pp. 110-122, Jan. 1998.

[5] D.P. Bertsekas, "Dynamic Programming and Optimal Control," *vols I and II*, Athena Scientific, Massachusetts. 1995.

[6] A. Dan, P. Shahabuddin, D. Sitaram and D. Towsley, "Channel Allocation under Batching and VCR Control in Video-On-Demand Systems," *J. Parallel and Distributed Computing (Special Issue on Multimedia Processing and Technology)*, Vol. 30, No. 2, pp. 168-179, Nov. 1995.

[7] A. Dan, D. Sitaram, "Multimedia Caching Strategies for Heterogeneous Application and Server Environments," *Multimedia Tools and Applications*, Kluwer Academic Publishers, Vol. 4 Number 3, pp. 279-312, May 1997.

[8] D. Eager, M. Vernon, and J. Zahorjan, "Optimal and Efficient Merging Schedules for Video-on-Demand Servers," *Proc. ACM Multimedia '99*, Orlando, FL, USA, pp. 199–202, Nov. 1999.

[9] B. Furht, R. Westwater, and J. Ice, "IP Simulcast: A New Technique for Multimedia Broadcasting over the Internet," *Journal of Computing and Information Technology*, Vol. 6, No. 3, pp. 245-254, Sept. 1998.

[10] M. Garey and D.S. Johnson, "The Rectilinear Steiner Tree Problem is NP-Complete," *SIAM J. Applied Mathematics*, Vol. 32, pp. 826-834, 1977.

[11] 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.

[12] T.F. Gonzalez, "On the Computational Complexity of Clustering and Related Problems," *Proc. IFIP Conference on System Modeling and Optimization*, New York, NY, USA, pp.174-82, Sept. 1981.

[13] K.A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," *Proc. ACM Multimedia '98*, pp. 191–200, Bristol, U.K., Sept. 1998.

[14] K.A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," *Proc. ACM SIGCOMM '97*, pp. 89–100, Sept. 1997.

[15] W. Ke, P. Basu, and T.D.C. Little, "Time Domain Modeling of Batching under User Interaction and Dynamic Piggybacking Schemes," *Proc. SPIE MMCN '02*, San Jose, CA, Jan. 2002.

[16] R. Krishnan and T.D.C. Little, "Service Aggregation Through a Novel Rate Adaptation Technique Using a Single Storage Format," *Proc. NOSSDAV '97*, St. Louis, MO, May 1997.

[17] R. Krishnan, D. Ventakesh and T.D.C. Little, "A Failure and Overload Tolerance Mechanism for Continuous Media Servers," *Proc. ACM Multimedia '97*, Seattle, WA, USA, pp. 131-142, Nov. 1997.

[18] S.-W. Lau, J.C.S. Lui and L. Golubchik, "Merging Video Streams in a Multimedia Storage Server: Complexity and Heuristics," *Multimedia Systems*, Vol. 6, No. 1, pp. 29-42, 1998.

[19] M. Leung, J.C.S. Lui and L. Golubchik, "Buffer and resource I/O Pre-Allocation for Implementing Batching and Buffering Techniques for Video-on-Demand Systems," *ICDE '97*, Birmingham, UK, Apr. 1997.

[20] J-.P. Nussbaumer, F. Schaffa, "Impact of Channel Allocation Policies on Quality of Service of Video on Demand over CATV," *Multimedia Tools and Applications*, Kluwer Academic Publishers, Vol. 2, Number 2, pp. 111-131, Mar. 1996.

[21] S.K. Rao, P. Sadayappan, F.K. Hwang and P. Shor, "The Rectilinear Steiner Arborescence Problem", *Algorithmica*, Vol. 7, pp. 277-288, 1992.

[22] S. Sheu and K.A. Hua, "Virtual batching: A new scheduling technique for video-on-demand servers," *Fifth International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, Apr. 1997.

[23] W. Shi and C. Su, "The Rectilinear Steiner Arborescence Problem is NP-Complete," *Proc. ACM/SIAM SODA '00*, San Francisco, CA, Jan. 2000.

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

[25] 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, Number 1, pp. 61-83, Jan. 1998.

[26] D. Venkatesh and T.D.C. Little, "Dynamic Service Aggregation for Efficient Use of Resources in Interactive Video Delivery," *Lecture Notes in Computer Science, Vol. 1018, (Proc. NOSSDAV '95)*, T.D.C. Little, R. Gusella, Eds., Springer-Verlag, pp. 113-116, Nov. 1995.

[27] P. Wegner, "Why Interaction Is More Powerful Than Algorithms," *CACM*, Vol. 40, No. 5, pp. 80-91, May 1997.

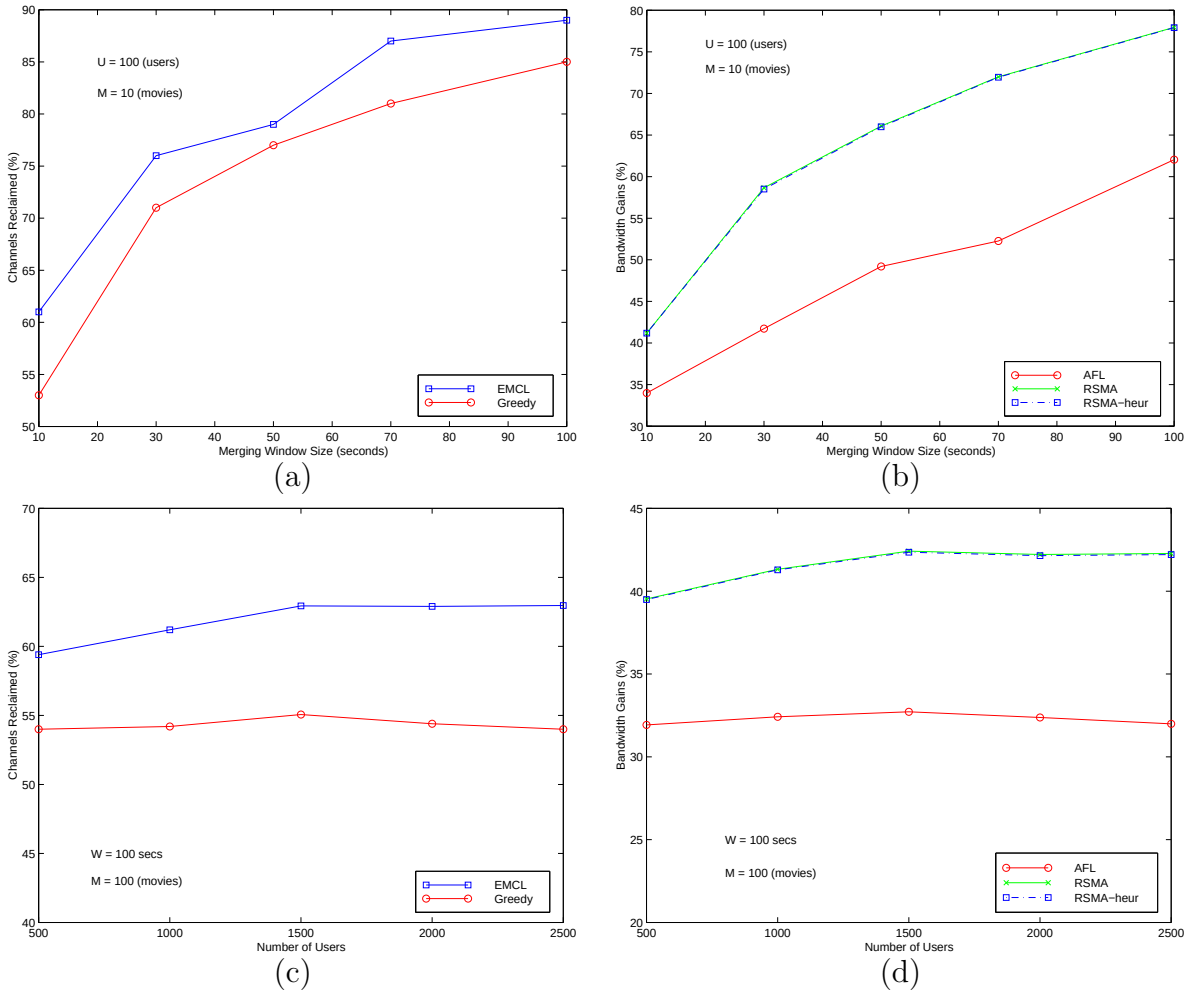


Figure 7: Static Snapshot Case

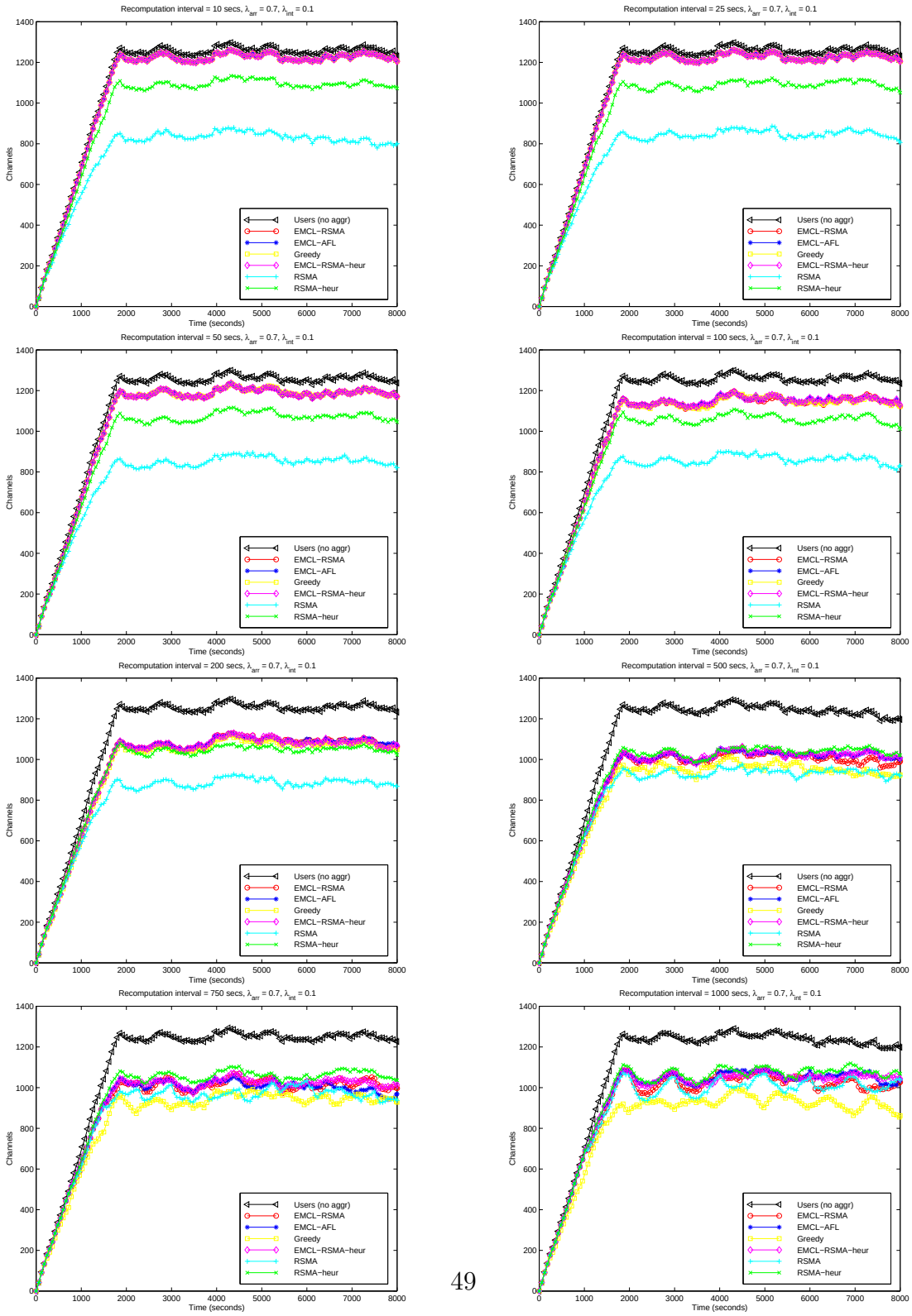


Figure 8: Dynamic Clustering in Steady State

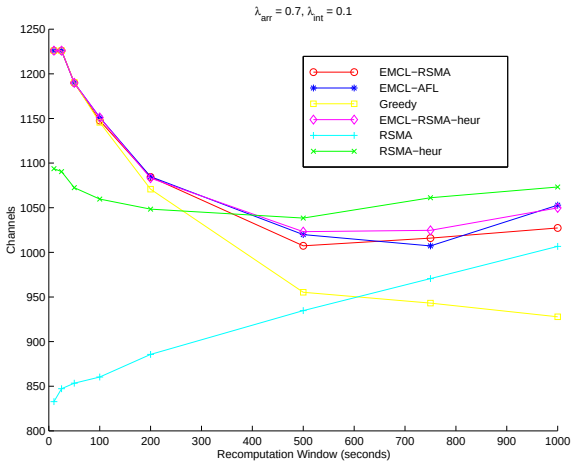


Figure 9: Dependence of Channel Gains on W

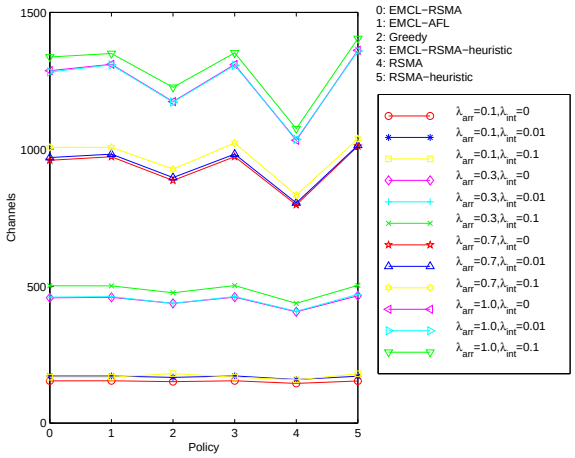
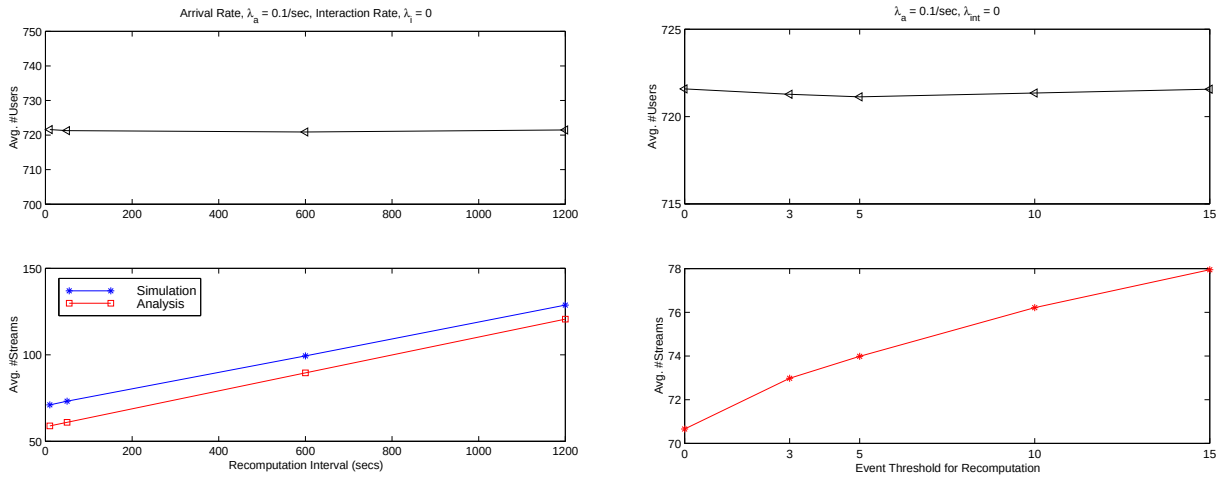


Figure 10: Comparison: With Best W for each Policy



(a) RSMA Slide vs. Recomputation Interval (b) Event Triggered RSMA vs. Event Count

Figure 11: Two Variants of RSMA

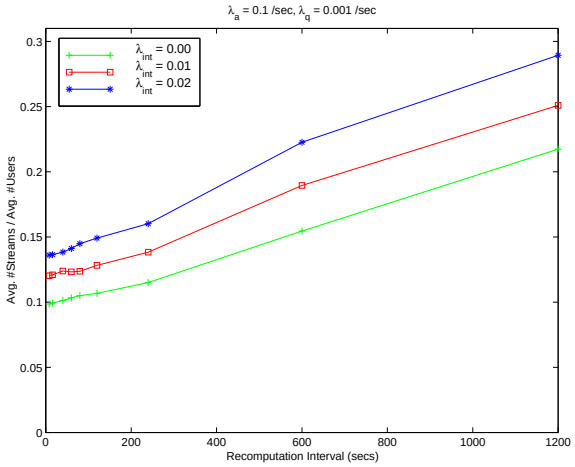


Figure 12: Effect of Varying Degrees of Interactivity