Optimal Stream Clustering Problems in Video-on-Demand[1]

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

Multimedia Communications Laboratory
Department of Electrical and Computer Engineering
Boston University
8 Saint Mary's St., Boston, Massachusetts 02215, USA
(617) 353-9877, (617) 353-6440 fax
{*pbasu,krash,tdcl*}*@bu.edu*

MCL Technical Report 04-22-98

Abstract – Supporting 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. Clustering of users by bridging the temporal skews between them is one such service aggregation technique. We present some recurrent problems in stream clustering and investigate optimal solutions with the main goal of minimizing average bandwidth. We also show that for dynamic interactive scenarios where streams can break away from clusters, given perfect prediction, the problem is isomorphic to the Rectilinear Steiner Minimal Arborescence (RSMA) problem; the complexity of which remains open and for which no polynomial-time algorithms are known. Because more general problems of interest are NP-Complete, we investigate and show results of performance evaluations of heuristic and approximate algorithms for this case. These algorithms are general across a large class of implementation architectures and aggregation techniques. Our simulations show that for a moderately interactive user population of 1250 at steady state, the RSMA algorithm is able to reclaim up to 33% of the bandwidth. Thus the static case optimal solution also performs as an excellent heuristic in the dynamic case.

Keywords: Dynamic service aggregation, clustering algorithms, content insertion, video-on-demand.

[1]A shorter version appeared in Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems (PDCS) 1998, (Special Session on Communications and Computing for Distributed Multimedia Systems), Las Vegas NV, October 1998. This work is supported in part by the National Science Foundation under Grant No. NCR-9523958.

1 Introduction

Supporting VCR-like interactions in true-video-on-demand requires the allocation of a channel to each interactive user. In the worst case, the system must be provisioned with a dedicated channel for each user.

Several service aggregation techniques for VoD have been presented in the literature. They include batching [3], rate adaptive merging [6], server caching [17], client caching or bridging [1] and chaining [16] (a limited form of distributed caching), content insertion [12] and content excision.

Aggregation schemes primarily fall into the two categories of clustering and caching. Clustering minimizes end-to-end bandwidth requirement by bridging the temporal skew between streams carrying the same content. This can be done in several ways: by rate adaptive merging [6], by content insertion [12] or by content excision. One can view stream clustering as a synchronization problem where the leading and trailing streams are out of "sync" and we can bridge the skew by changing the relative content progression rates. One can also view this as a control problem where the skew is viewed as an error which is to be minimized, for example, by setting the leading stream back via content insertion.

Rate adaptive merging of two streams can be achieved by accelerating the trailing stream by about 7%, towards the leading stream until both the streams are at the same position in the program [11]. Then all the users on those two streams can be served by a single stream.

Caching schemes minimize the number of streams required on the bottleneck path (typically storage) by buffering the interval between streams and playing some streams out of the buffer. It is also desirable to reduce the buffer requirement, which is dependent on the lengths of the intervals 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.

Due to the long playout time and high density access characteristics of continuous media, we see that caching and clustering are related. A continuum of aggregation schemes which lie between pure caching and pure clustering exist. Tsai and Lee [15] explore this relationship in the near-VoD scenario.

Therefore, we can extend solutions for clustering to buffer management and vice versa. In this paper, we restrict our attention only to clustering schemes. Though presented in a

VoD setting, a number of these results are applicable to other settings like personalized news delivery over polychannel architectures as in Gifford [5].

The aforementioned schemes take advantage of the fact that typical content access distributions are skewed. During peak demand, there is a high density of access – a large number of users request the same content but with small temporal skews. Aggregation schemes are attractive in VoD systems since peak demands far outstrip available bandwidth.

Since resource sharing by aggregating users is a promising paradigm for VoD systems supporting large user populations, it is useful to study the 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 paper, we make a systematic approach to formulate these clustering problems and examine if optimal solutions exist. 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. The other issue is to explore how far the gains from an optimal solution in the static case, where the clusters cannot break away once they are formed are being held in dynamic scenarios. Our performance results can be readily applied to the related capacity planning and design problem – given an interaction plus arrival rate and distribution, what should the design capacity be, in number of streams, of a video network which actively uses service aggregation?

We transform the static clustering problem (with the objective of minimizing average bandwidth) to the RSMA-slide problem[2]. We show that by periodically recomputing the RSMA-slide with small a period, we can reclaim up to 33% of channel bandwidth when there are 1250 interacting users in the system at 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

[2]An RSMA-slide is essentially an optimal binary tree.

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.

The rest of the paper is organized as follows. We formulate the clustering problems in Section 2 and analyze optimal algorithmic solutions. In Section 3 we present the results of simulations of heuristic and approximate algorithms for clustering. In section 4 we conclude with a set of recommendations for algorithms for use in stream clustering in interactive VoD systems.

2 Stream Clustering Problems

In this section, we characterize the space of video stream clustering problems by formulating a series of sub-problems. We will demonstrate that problems of interest are open; however, the formulation lends itself to heuristic solution approaches. In the next section, we present effective heuristic solutions to these problems.

2.1 Clustering Under a Time Constraint

Let us consider the problem of recovery from overload resulting from front-end processor failure in a clustered[3] video server. The objective is to recover the maximum number of channels possible within a finite time budget, by clustering streams. We assume that interactions are not honored during the recovery period since the system is already under overload.

Maximizing channel recovery is desirable since 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 like advertisements. This period can be exploited to reconfigure additional resources. However reconfigurability adds substantially to system costs and additional resources may not be generally 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 can be found in [12]. 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

[3]Here "clustered" refers to a server architecture where multiple front-end processors enhance delivery throughput of content from shared storage.

by starting from the next stream ahead until the trailing stream is reached. A more general class of uni-dimensional clustering problems has been shown to be polynomially solvable [7].

Three points are worth noting here. Firstly, the algorithm presented in [12] can be applied when both content insertion and rate adaptation are used. A little thought shows that 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 minimization constraint and dynamicity in the following sections.

2.2 Clustering to Minimize Bandwidth

Clustering to minimize the average bandwidth requirement, in number of channels, is more applicable from a service aggregation perspective. As a special case of this problem let us consider the case in which we have a snapshot of stream positions at a given instant and we wish to construct a merging schedule that uses the minimum average bandwidth. Further let us ignore arrivals, exits and break-aways due to interaction. This problem has been considered by Lau et al. [10] for a heuristic solution only.

We find that this problem can be readily transformed into a special case of the Rectilinear Steiner Minimal Arborescence (RSMA) problem, which is defined by Rao et al., [14]:

> Given a set N of n nodes lying in the first quadrant of E^2, find a minimum length directed tree (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.

Unfortunately the complexity of the RSMA problem is still open and Rao et al. [14] present a

$\mathcal{O}(n \log n)$ approximation algorithm which gives a solution within twice the optimal. Known exact algorithms have an exponential worst-case complexity (see Leung et al. [8]).

The transformation is illustrated in Figure 1 with a minor difference that an *up arborescence*[4] is used instead. In the figure on the left, the purple dots denote the stream positions at the time instant corresponding to the beginning of the snapshot. Each stream has a choice of either going at a normal speed (denoted by red dotted lines) or at an accelerated speed (denoted by blue dotted lines). In the optimal tree, the absolute trailer has to go fast and the absolute leader has to go slow all the time and the two lines corresponding to them will meet at the root. In Steiner tree terminology the purple points ($\in N$) are called *sinks*. The RSMA tree is denoted in the figure on the right as a rooted directed tree (the arrow directions have been reversed for a better intuitive feel).

Note that the transformation leads to a special case of the RSMA problem where the sinks form a *slide*, which is a configuration where there is no directed path from one sink to another. For this special case, the RSMA is an optimal binary tree and can be computed using a $\mathcal{O}(n^3)$ dynamic programming algorithm [14]. We call it the RSMA-slide in this paper. Therefore, we conclude that the reasoning given by Lau et al. in [10] is incorrect, since the static clustering problem has an optimal polynomial time solution.

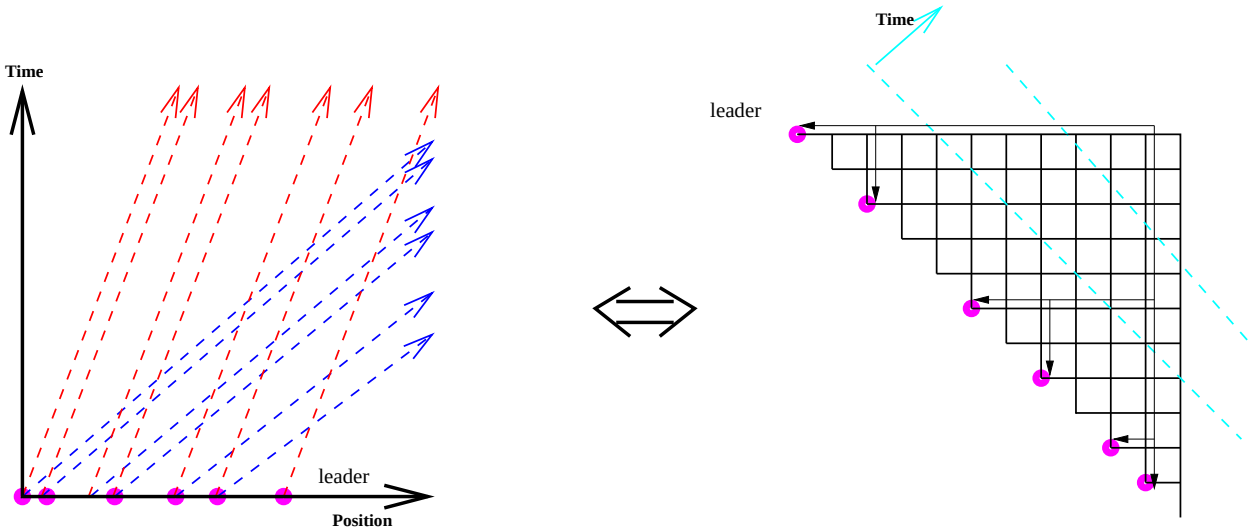

Figure 1: Minimum Bandwidth Clustering

Aggarwal et al., [2] present the optimal binary tree formulation to this problem and also provide a $\mathcal{O}(n^3)$ dynamic programming algorithm. Although optimal binary trees are a

[4]A directed tree with arcs directed towards the root.

direct way to model this problem, the RSMA formulation offers more generality as we shall explain in later sections.

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. In a dynamic scenario, where cluster re-computation is performed on periodic snapshots, this technique can speed up computations. 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 are reported in Section 3.

2.3 Clustering with Arrivals

Let us introduce only stream arrivals to the preceding problem and ignoring stream exits and interactions. Stream exits can occur due to the ending of a movie or a user quitting from a channel which has only one associated user. Suppose we have a perfect predictor of new stream arrivals, the problem again transforms to the RSMA-slide formulation. Although the sinks do not lie on a straight line, they still form a slide. E.g., the purple points in Fig. 2 form a slide. Therefore, the RSMA-slide algorithm of Section 2.2 can be used here.

Alternately, 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.

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, advance reservations for example, were available, then by using the exact RSMA-slide algorithm, we can do better.

2.4 Clustering with Interactions

Here we show how a highly restricted version of the problem with certain unrealistic assumptions is transformable to the RSMA problem. We assume the jump type interaction model i.e., a user knows where to fast forward or rewind to and resumes instantaneously. Also assume that the leading stream does not exit and that interactions do not occur in

singleton streams (which are alone in their clusters). Above all this, if we assume that a perfect prediction oracle which predicts the exact jumps, exists, we can see that the problem of merging all the streams using minimum average bandwidth is isomorphic to the RSMA problem, which is open and thus has no polynomial time solution.

The main point of the above argument is that even with all the above unrealistic assumptions about streams, the problem is open; the general problem is harder and is also open from the optimality considerations. As we will see in Section 3, certain heuristic approaches based on the static RSMA algorithm perform reasonably well in practice.

2.5 Clustering with Arrivals, Exits and Interactions

In this subsection, we consider optimal stream clustering with new stream arrivals and VCR interactions. Let us consider an oracle which can perfectly predict arrivals and interactions over large periods. Fig. 2 shows a transformation to the RSMA problem in the general case. The purple points on the horizontal line at the bottom denote the predicted arrivals and the green points in the interior of the grid depict predicted interactions. Also, the assumptions of Section 2.4 are assumed to be valid. The problem is to find a schedule which merges all streams, taking into account all the predicted arrivals and interactions, and consumes minimum bandwidth during the process. We can easily see that solving this problem is again isomorphic to solving the general RSMA problem which unlike the RSMA-slide algorithm does not have a polynomial time solution.

Stream 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. 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 at least as hard as the RSMA problem and thus no optimal solution exists.

An approximation algorithm for RSMA which performs within twice the optimal is available using a plane sweep technique. But in a practical situation where there is no perfect prediction oracle, knowing the arrivals and interactions beforehand is not feasible. Therefore one doesn't have much choice other than taking periodic snapshots of the system. A point to be noted is that there is no optimal window for recomputing optimal clustering based on periodic snapshots because the problem doesn't have an optimal polynomial time solution even with perfect prediction.

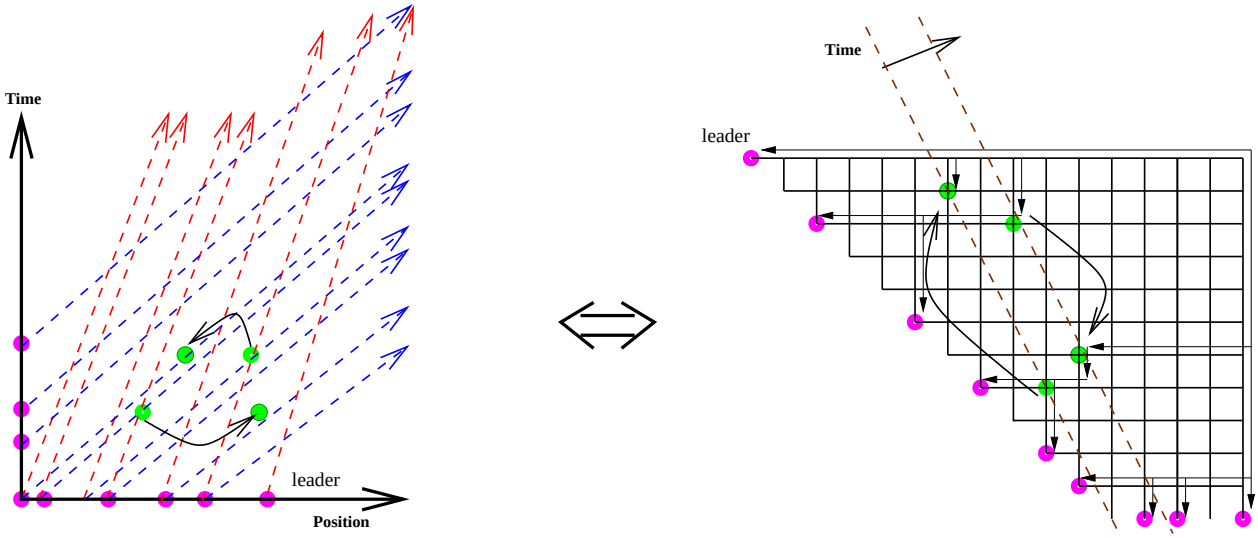

Figure 2: Clustering under Perfect Prediction

Also note that causal event driven cluster re-computation 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. It is more advisable to seek stochastic bounds since the worst case bound cannot be guaranteed better than twice optimal with perfect prediction unless the RSMA problem is solved in polynomial time.

Note that an algorithm with time complexity $\mathcal{O}(n^3)$ may be expensive for an on-line algorithm and it is worthwhile comparing it with its approximation algorithm which runs faster, producing results within twice the optimal in the worst case. In the simulations section (Section 3), we compare both the algorithms under dynamic situations. Rao et al. [14] give a polynomial-time approximation algorithm based on a plane-sweep technique which gives an RSMA within twice the optimal. We present this algorithm here in pseudocode form (Fig. 3), which we use later for simulations.

When the algorithm terminates, we get the heuristic tree. Since we can store the points in a *heap* data structure, we can perform Step 3 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 [19] suggests that interaction is more powerful than algorithms and such scenarios may be difficult to model using an algorithmic framework.

Algorithm **RSMA-heuristic**($\mathcal{S}$:*sorted stream list*)

1. Place the points in $\mathcal{S}$ on the RSA grid
2. `while` ($\mathcal{S} \neq \phi$)
3. Find 2 nodes $p, q \in \mathcal{S}$ such that
   $\|parent(p,q)\|$ is maximum
   (Note: $\|(x,y)\| = x + y$)
4. $\mathcal{S} \leftarrow \mathcal{S} - \{p,q\} \cup parent(p,q)$
5. Append the branches $parent(p,q) \rightarrow p$ and
   $parent(p,q) \rightarrow q$ to the RSA tree.
6. `end`

Figure 3: Approximation Algorithm for RSMA

2.6 Harder Constraints and Heuristics

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

- Limits on total content insertion permissible per customer
- Limits on total duration of rate adaptation per customer
- Maximum duration and frequency of content insertion per customer
- Critical program sections where content-insertion is prohibited
- Critical program sections where rate adaptation is prohibited
- Frequency of rate changes per customer
- Continuous rather than "jump" interactions

Some generalizations involve multiple customer classes or multiple adaptation rates with different costs although the latter is impractical from a retrieval and delivery perspective.

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 some heuristic approaches which are worthy of experimental evaluation.

- **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-driven** Take a snapshot of the system whenever the number of new arrivals or interactions exceeds a given threshold. As a refinement the count can do this on a per-movie basis, that is, recompute for that movie if the number of arrivals and interactions exceeds a threshold. Recomputing on every interval will lead to sub-optimal results.
- **Adaptive** Adapt the re-computation period based on interactivity and the state of the system.
- **Predictive** Predict arrivals and interactions based on observed behavior. Compute the RSMA based on predictions.
- **Policy iteration** Based on observed system behavior over a large period of time, compute the best policy for each set of circumstances. Choose the best policy for the current circumstance.

3 Simulations

In this section, we describe our simulations of the clustering and merging algorithms which we have described in the first half of the paper, and also compare them with some heuristic algorithms which will be described in this section. We describe simulations of algorithms and heuristics for only rate adaptation in this section.

3.1 Objective

Some of the algorithms described in Section 2 are optimal in the static case i.e in absence of user interactions. Also, no optimal algorithms are known for the dynamic scenario in which users are arriving into the VoD system, interacting with it and departing from it. However, the behavior of such a system upon the application of our clustering and merging techniques can be studied by simulations. In these simulations, we shall study the gains offered by our ensemble of techniques, under varying conditions. We will also attempt to answer the following questions:

- In the non-interactive situation, which is the best policy?
- Which is the best policy 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?

3.2 The Simulation Setup

We treat our simulation as a discrete-time control problem. As it has been outlined in Section 2, there are various ways in which we can control the system: apply control with a fixed frequency, apply control with a fixed event frequency or adapt control frequency depending on the event frequency, which is the hardest to implement. In this work, we have considered only the first method of applying control periodically and have left the remaining as a part of future work.

The 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 the exponential probability distribution. In this study, 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 in every re-computation period, the simulation driver conveys the status of each stream to the clustering unit and queries it after every simulation tick to get the status of the streams 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 gets the status of all running streams from the simulation driver, 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, it supports querying on the status of any running non-interacting stream by the simulation driver after every simulation time unit.

3.3 Assumptions and Variables

The variables in our simulations are the following: number of movies, M; movie popularities obey Zipfian distribution, length of a movie, L, which is 30 minutes in these simulations, recomputation window size, W, mean movie request rate, λ_{arr}, mean interaction rate, λ_{int}, and mean interaction duration, λ_{dur}, which is taken to be 5 sec for simplicity in our simulations; all λ's are exponentially distributed.

We have simulated three types of interactions: *fast forward, rewind* and *pause*. λ_{int} is a individual parameter which 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.) happens in the system once every 50 seconds. For simplicity we have kept λ_{int} the same for each type of interaction.

The size of the complete user pool does not play an important role in our simulations for the dynamic scenario because even in real life, all the users in the complete pool are never in the system at the same time. Only λ_{arr} and λ_{int} determine the number of users in the system at a given time. So, we have assumed a reasonably large size for the user pool at $U = 2500$. Also, in this work, we do not study the probability of *blocking* of a requesting user as we assume that the capacity of the system is greater than the number of occupied channels.

We have assumed that the effective frame rate of a normal speed stream is 30 fps and that of an accelerated stream is 32 fps; this can be achieved in practice by dropping a B-picture from every *group of pictures*, keeping the streaming and playout rates at 30fps. Also, a user may fast forward/rewind at $5X$ normal speed. Finally we have kept our simulation granularity at 1 sec and the simulation duration at 8000 seconds.

We investigated the channel and bandwidth reclamation rates for each clustering algo-

rithm and compared the results (Section 3.5).

3.4 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 making all the trailers in a cluster 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 cheaper than the RSMA-slide algorithm.

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

RSMA-heuristic The RSMA-heuristic algorithm is applied to the entire set of streams (over the full length of the movie) periodically 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 made if the immediate leader is a normal speed stream. If that's true 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 [6] apart from the fact that it re-evaluates the situation at every interaction event too.

The first three policies release the same number of channels at the end of a certain period because they use the EMCL algorithm for creating merge-able clusters. But, the amount of bandwidth saved in the merging process is different in each case. EMCL-RSMA is proved to be optimal over a given merging window, so it saves the maximum bandwidth. EMCL-AFL is a mere heuristic and could perform badly in many situations. EMCL-RSMA-heuristic on the other hand uses an approximation algorithm for the RSMA-slide that guarantees a solution within twice the optimal. [14] Although our problem falls within the *slide* category

and has an $\mathcal{O}(n^3)$ exact solution, we wanted to experiment with the approximation algorithm which runs in $\mathcal{O}(n \log n)$ time.

3.5 Results and Analysis

In this subsection, we study clustering (channel reclamation) gains and merging (bandwidth reclamation) gains due to the previously discussed algorithms.

We have classified the simulations into these two categories:

3.5.1 Snapshot Case

This is applicable in overload situations, i.e., at a certain point of time when the server detects an overload, it wants to release as many channels as possible in a static merging window (i.e., there are no user interactions in this period), consuming minimum bandwidth at the same time. First, we vary the window size for a fixed number of users. In Fig. 4(a), we can see that the clustering gain increases as W is increased. We can also see that EMCL outperforms GreedyMerge over any static window of size W. In Fig. 4(b), we have compared the bandwidth gains due to the three merging algorithms that we have talked about earlier. We observe that merging gain increases with increase in W. We also observe that 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 situations. (the reason for this may be attributed to the parameters of the particular inter-arrival distribution) Although cases can be constructed where the heuristic produces a result which is twice the optimal value, in most practical situations it performs as well as the exact algorithm at a lower computational cost.

Next, in Fig. 4(c), we vary the number of users for a fixed window size (100 secs) and compare 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 seems to increase as U increases. In Fig. 4(d), we can again see 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 seem to remain constant as U increases.

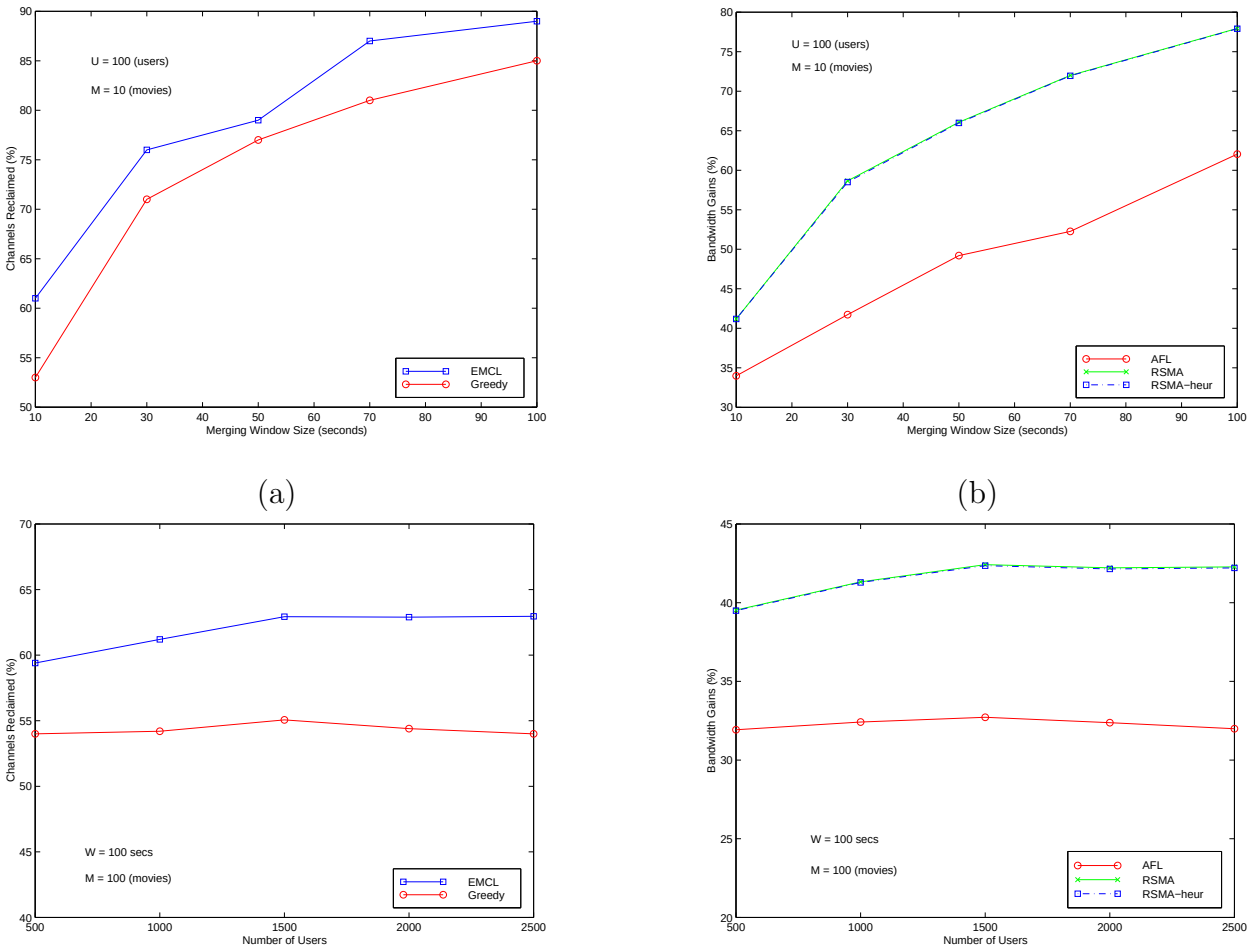

Figure 4: Snapshot Case

Table 1: 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	$\lambda_{arr} = 1.0, \lambda_{int} = 0$	$\lambda_{arr} = 1.0 \lambda_{int} = 0.01$	$\lambda_{arr} = 1.0, \lambda_{int} = 0.1$

3.5.2 Dynamic Case

This is the general scenario where the users come into the system, interact with the system and leave. Here, we simulate over time equivalent to about $4\frac{1}{2}$ lengths of the movies.

First, 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 have considered 12 different arrival-interaction patterns as shown in Table 1.

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 1000 seconds. Fig. 5 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 1250 channels after the steady state has reached thus giving 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 1250 streams.

For smaller W, other policies on the other hand give much lesser 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 re-computation interval and advances streams according to the *RSMA tree* computed from that snapshot. But in this situation where the user arrival rate is moderately high and the interaction rate is high too, for high values of W, a lot of streams come in and interact between two consecutive snapshots and thus the gains due to RSMA drop. On the other hand, GM tries to reevaluate the situation at every arrival and interaction event, so it performs very well.

Our hypothesis was that different policies will perform at their best for different values of W and indeed, we found that to be true. Fig. 6 shows how the channel gains vary with the value of the re-computation window. For consistency, we have considered 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 paucity of space. In the steady state, the system has 1250 users on average. For small values of W ($\leq 500\,secs$), EMCL based schemes and GM don't 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 better than the EMCL based algorithms. GM outperforms all algorithms, including RSMA for

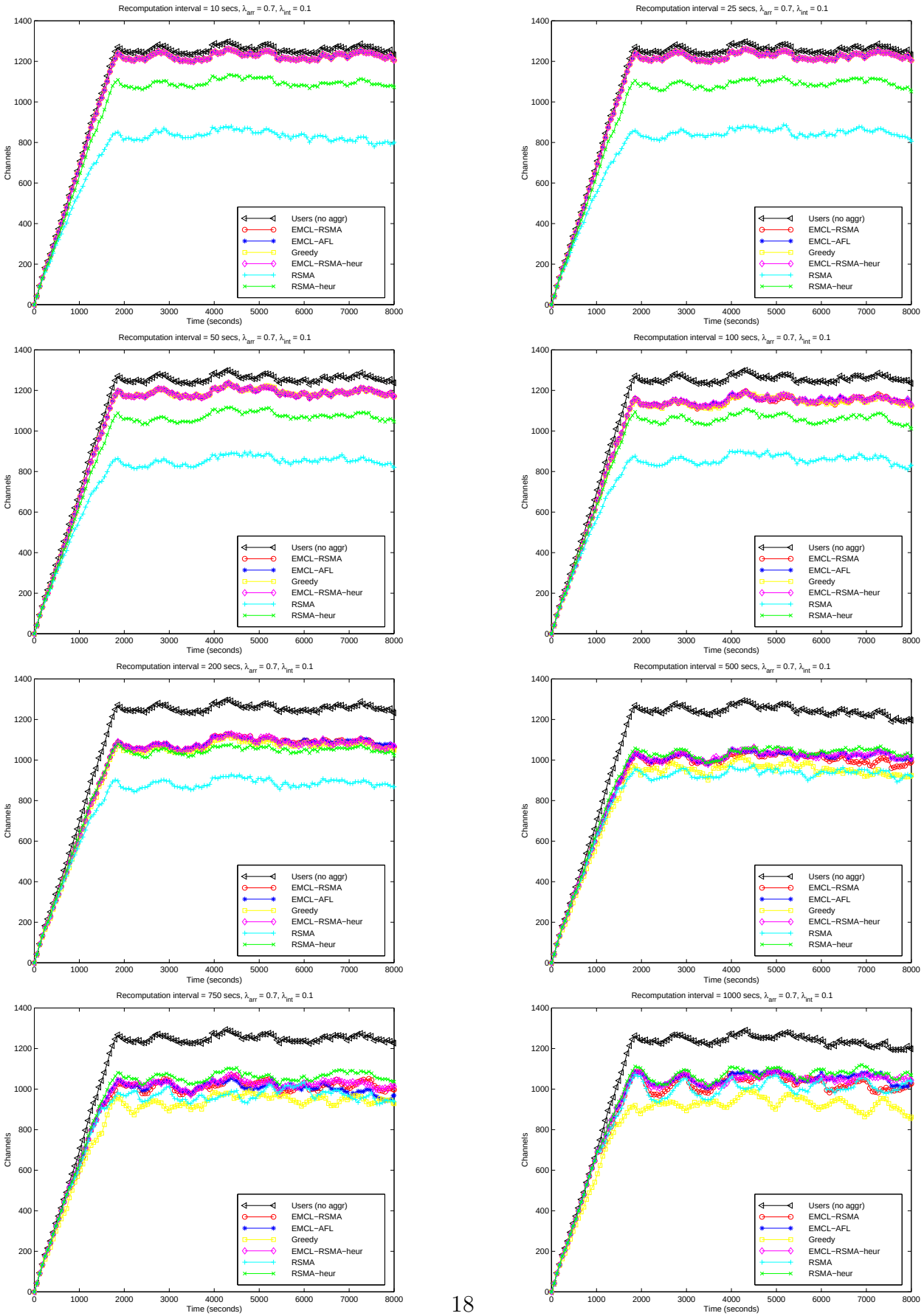

Figure 5: Clustering in Steady State

$W \geq 750$ 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 very scalable solution under heavy rates of arrival and interaction although it may perform better than the other algorithms in such situations.

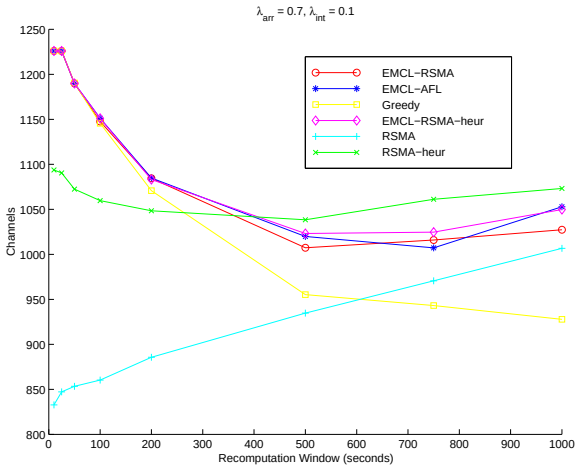

Figure 6: Dependence of Channel Gains on W

As mentioned in the last paragraph, each policy attains best results for different values of W. RSMA performs best for $W = 10\,sec$; GM performs best for $W = 1000\,secs$ and the EMCL based algorithms perform best for $500 \leq W \leq 750$. Fig. 7 plots the best channel gain achieved by a policy for every class of traffic (see 1). In the figure, we can see 4 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 are $180, 540, 1250$ and 1800 respectively. We can easily see that the mean clustering gains increase significantly with increase in the arrival rate.

For each of the 4 sets, we observe that 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, we see that RSMA emerges as the best overall policy. We can also see that as $\lambda_a rr$ increases, the gap between RSMA and the other algorithms widens.

We can also see that 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 (since EMCL hasn't been applied to the stream cluster), so the sub-optimality of the heuristic shows up.

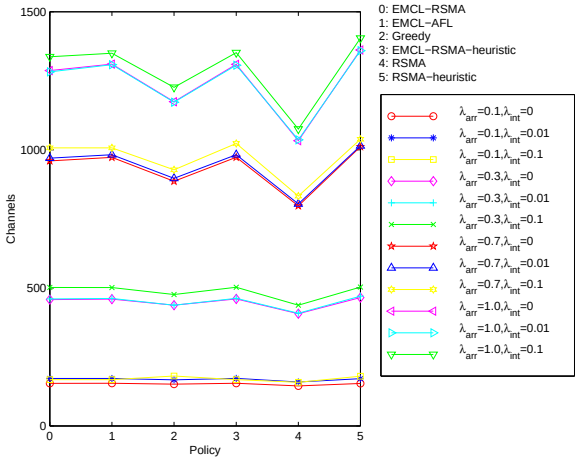

Figure 7: Comparison: With Best W for each Policy

From the above simulations, we can conclude the following:

- EMCL-RSMA is the optimal policy in the static snapshot case as it can reclaim maximum number of channels with minimum bandwidth usage during merging. Thus it is the ideal policy for server overloads.
- For dynamic scenarios, if the number of users in the system is not too large ($\sim$ 2000 watching 100 movies) and the rate of interaction is moderate (500 interactions in an hour), periodically invoking RSMA with low periods gives the best results.
- In realistic VoD settings, 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 Conclusions and Future Work

In this paper, we described various optimality criteria arising in stream clustering problems in video on demand. We gave an optimal solution for the "static clustering under a time budget" problem which has $\mathcal{O}(n^3)$ running time. We also showed that a restricted version of the dynamic problem (with user arrivals and interactions) is transformable to RSMA which is open, i.e., it is not known to be in $\mathcal{P}$ or $\mathcal{NP}$.

We showed full scale simulations of the dynamic scenario with users entering the system, interacting with the system and leaving the system. In the simulation setup, we ran six different aggregation policies and compared their clustering gains in the steady state. We observed that periodic invocation of the RSMA-slide clustering algorithm with small periods produces best results under moderately high rates of arrival and interactions.

In future work, we propose to investigate the feasibility and complexity of distributed clustering schemes and possible clustering together 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.

An 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.

In this paper, we did not study blocking of users due to depletion of channels but in any system with a scarce resource, blocking and service denial is inevitable and its probability has to be minimized. It will also be interesting to study how adaptive invocation of the clustering algorithms performs under situations similar to those described in this paper.

To summarize, the main contributions of this paper are the following : (i) general formulation of the stream clustering problem and solution approaches (ii) inclusion of interactions in the model and (iii) performance of approximate and heuristic algorithms in a dynamic scenario.

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 Journal on Selected Areas in Communication,* 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. SIGMETRICS '96: Conference on Measurement and Modeling of Computer Systems,* Philadelphia, PA, USA, pp. 200-209, May 1996.

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

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

[5] D.K. Gifford, "Polychannel Systems for Mass Digital Communication," *Communications of the ACM,* Vol. 33, No. 2, pp. 141-151, Feb 1990.

[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] T.F. Gonzalez, "On the Computational Complexity of Clustering and Related Problems," *Proceedings of the 10th IFIP Conference on System Modeling and Optimization,* New York, NY, USA, pp.174-82, Sep 1981.

[8] K-.S. Leung and J. Cong, "Fast Optimal Algorithms for the Minimal Rectilinear Steiner Arborescence Problem," *Proc. 1997 IEEE International Symposium on Circuits and Systems,* Hong Kong, Vol. 3, pp. 1568-1571, Jun 1997.

[9] 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," *Thirteenth International Conference on Data Engineering,* Birmingham, UK, Apr 1997.

[10] 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.

[11] R. Krishnan and T. D. C. Little, "Service Aggregation Through a Novel Rate Adaptation Technique Using a Single Storage Format", *Proc. 7th Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, St. Louis, MO, May 1997.

[12] R. Krishnan, D. Ventakesh 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.

[13] 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, Issue 2, pp. 111-131, Mar 1996.

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

[15] 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.

[16] 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.

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

[18] 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. of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video),* T.D.C. Little, R. Gusella, Eds., Springer-Verlag, pp. 113-116, Nov 1995.

[19] P. Wegner, "Why Interaction Is More Powerful Than Algorithms," *Communications of the ACM,* Vol. 40, No. 5, pp. 80-91, May 1997.