Sharing Video on Demand ⋆

Amotz Bar-Noy [a,1] Juan A. Garay [b,2] Amir Herzberg [c]

[a] *Computer and Information Science Department, Brooklyn College, 2900 Bedford Avenue, Brooklyn, NY 11210.* amotz@sci.brooklyn.cuny.edu.

[b] *Bell Labs – Lucent Technologies, 600 Mountain Avenue, Murray Hill, NJ 07974.* garay@research.bell-labs.com.

[c] *IBM Haifa Research Center, Room 418, 2 Weizmann Street, Tel-Aviv 61336, Israel.*

Abstract

We formulate the problem of Video on Demand (VOD) from a resource allocation perspective. We introduce the *decision element* into a movie vending environment, which complements the current approaches. In contrast with the traditional resource allocation problems (e.g., machine scheduling), the problem possesses the distinctive *batching* property, which stands for the feasibility of several requests being served by one resource.

First, we investigate the problem in an *on-line* fashion, namely, having to accept or reject a request for a movie without any knowledge about future requests. We show upper and lower bounds on the *competitive ratio* of on-line movie scheduling algorithms for a variety of scenarios depending on the notification time. In particular, when the length of the movie and the notification time on requests are linearly related, we present a simple competitive algorithm.

Second, we propose a new approach – *Shared* VOD – which gives a more balanced functionality/price tradeoff than the two popular approaches Full VOD and Near VOD. In this approach, several users may share the same channels, but the movies are selected by the users themselves.

In this framework, we evaluate a variety of on-line channel allocation algorithms under "typical" conditions. We compare these strategies to the off-line strategy, and show that the algorithm we introduce substantially outperforms the rest.

Key words: Video on demand, resource allocation, on-line algorithms, competitive analysis.

1 Introduction

The technology for interactive home video entertainment has been actively developing. For example, hybrid networks that enable multi-media connections are being studied as a step toward all-digital video networks [10], as is the problem of bandwidth allocation strategies for combined analog/digital transmission of data over a CATV system [9]. Experiences and services trials are being performed, for both *Near* `VOD`, which consists of broadcasting predetermined movies at fixed time intervals (e.g., every 15 min) [13,17], as well as for *Full* `VOD`, where each user has a dedicated channel.

Most published works and on-going projects seem to share a common view on the basic architecture for `VOD` services, as depicted in Figure 1. The movies are stored in a central *video server*, which may be connected to other servers over a high bandwidth WAN. The *video server* is connected by a high-capacity fiber link to local distribution centers (hubs), from which coax cables are used to broadcast to the households. Either analog or digital transmission could be used from the server to the hubs and to the homes, where digital transmission may offer higher quality and capacity (by compression) but would require new, powerful set-top boxes. Alternatively to the coax cables, the hubs and the homes could be connected by phone lines using special coding and compression technology. With some reservations, our basic model and analysis apply to both analog and digital transmission, over either coax cable or telephone lines. [3]

⋆ Parts of this paper appeared as "Adaptive Video on Demand" in *Proc. 3rd Annual European Symp. on Algorithms*, Corfu, Greece, September 1995 [3], and parts were presented at the *Workshop on Algorithmic Aspects of Communication*, Bologna, Italy, July 1997.

[1] This work was done while the author was at the Electrical Engineering Department of Tel-Aviv University, Tel-Aviv, Israel, and supported by the Consortium for Broadband Communication administered by the Chief Scientist of the Israeli Ministry of Commerce and Industry.

[2] Work done while the author was at the IBM T.J. Watson Research Center, Yorktown Heights, NY, and visiting the Center for Mathematics and Informatics (CWI) of the Stichting Mathematisch Centrum, Amsterdam, The Netherlands. While at CWI, supported by NWO through NFI Project ALADDIN under contract number NF 62-376 and a SION grant, and by the European Union through NeuroCOLT ESPRIT Working Group No. 8556.

[3] Indeed, as a starting point, in this paper we simplify the implementation issues somewhat and concentrate on the scheduling problem, with the hope that we will still capture the insights for the right solutions. For example, if the transmission is digital, then we assume it to be CBR ("Constant Bit Rate"), and that the bit rate does not change from one video to another [14]. A more practical implementation would probably use a piece-wise linear CBR [20].

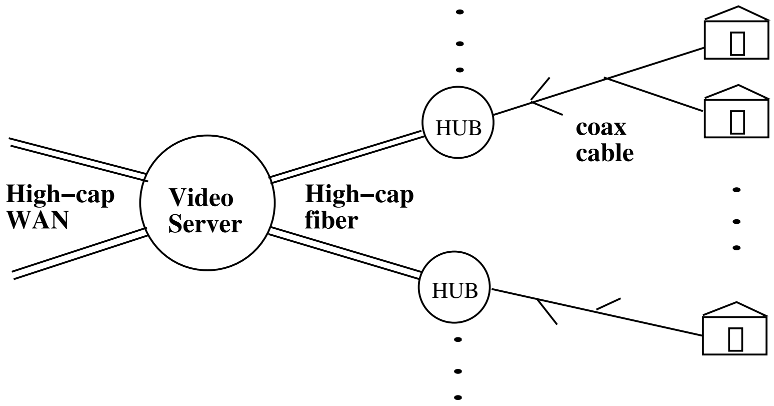

Fig. 1. Video-on-Demand architecture

In this architecture, there appear to be two major bottlenecks:

(1) The limited number of broadcast channels available on the access network (e.g., the coax cable). It appears that, for economic reasons, it may be desirable to put a much larger number of homes on the same broadcast cable. For example, the number of channels per coax may be about 100, while the number of concurrent users may range from 1000 to 5000.
(2) The server's *fan-out*, i.e., the number of movies which the server is able to transmit concurrently (see, in particular, [12]).

Although 1 is bound to improve with the advent of newer communication technologies, per our remark above, 2 seems to be a long-term problem, due to economic and business reasons. [4]

Previous works and current implementations attempt to achieve, under these constraints, *Full Video on Demand* and/or *Near Video on Demand*:

- **Full Video on Demand:** Whenever a user's request for a movie arrives, it is immediately served, provided there is capacity available (at the network, as well as at the server); otherwise the request is rejected.
- **Near Video on Demand:** A fixed set of movies is played regularly, at fixed time intervals.

Much attention has been given to the hardware requirements for such designs, and the quality of service, due to the bottlenecks mentioned above. It is immediate that either policy may poorly utilize the available resources. Namely, a considerable number of feasible distribution of requests may cause the first approach to tie up the distribution resources, while the latter provides—by definition—a limited service, and a bad decision directly translates into re-

[4] Again, a more thorough characterization of the server's behavior, which would take into account the variable time for retrieval of the data, is beyond the scope of this paper.

source wastefulness. The two approaches are shown in Figure 2.

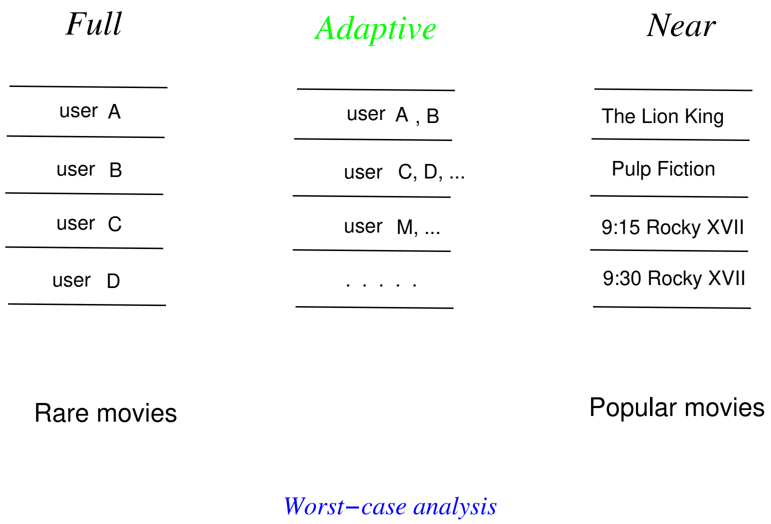

Fig. 2. VOD approaches

The purpose of the current paper is to investigate the intermediate terrain that lies between the two extreme policies. Indeed, we introduce the *decision element* into the video service architecture, which leads to the concept of:

- **Adaptive Video on Demand:** Upon arrival, a movie request is accepted and served (possibly with some delay), or rejected. The decision as to whether accept or reject (and the amount of delay) is made by a scheduling algorithm.

Note that the scheduling algorithm might specify that the request be rejected, even though there might be channels that are currently not being used. Intuitively, the goal of such a channel assignment algorithm is to maximize the "*sharing*" effect, i.e., the number of users assigned to the same channel.

The approach is also depicted in Figure 2. To the best of our knowledge, this is the first attempt so far to tackle VOD from an optimization perspective. Namely, the purpose of this paper is to optimize the decisions on which movie requests to accept, when to deliver them, and which requests to reject. We therefore try to maximize the utilization of a (limited) resource being shared by a number of users. Our motivation is to complement approaches based on predicting demand for movies (i.e., Near VOD), which are best suited for popular movies, but inappropriate for more unusual selections. In practice, we would expect the use of a mix of the approaches, with Near VOD taking care of the "hot," recent releases, and Full VOD providing the more rare selections, together with the appealing control functions—at a price. Adaptive VOD would cover the middle ground. We envision an implementation in which after a request for a movie is received, this movie is advertised so as to attract more requests for the same movie. We elaborate more on this in the sequel.

1.1 On-line algorithms

The nature of the case of unusual movie selections is such that it is always possible to "second guess" decisions made in the past. In other words, a decision made previously to accept a request, may have been wrong from the service provider point of view, because it may cause a channel to be used up, thus preventing a subsequent more "valuable" set of requests (for example, for a more popular movie) to be rejected. Thus, the *on-line* nature of the problem, i.e., the fact that decisions are to be made when requests for movies arrive into the system without knowledge of future requests, might lead to significantly lower efficiency than would have been possible with full *off-line* knowledge of the entire pattern of request arrivals. This motivates the emphasis we put on this issue. Namely, in this paper we start by investigating movie scheduling mechanisms in an *on-line* fashion. A second motivation for tackling a new problem with this type of analysis is the hope of achieving a deeper, meaningful understanding of the problem. Indeed, this has allowed us to come up with a `VOD` scheme (see Section 4) that we believe has the potential of practical realization.

We evaluate on-line algorithms in terms of their *competitiveness* [18]. An algorithm is said to be k-competitive, if its performance on *any* sequence of requests is within a factor k of the performance of any other algorithm on the same sequence, including the *off-line* algorithms for the problem—which can "foresee" all future requests. We remark that this on-line perspective is tantamount to a worst-case analysis; however, a main virtue of a good competitive mechanism is its **robustness**, i.e., its ability to provably behave well under all possible circumstances.

It is sometimes convenient for the analysis to describe things in terms of a game between a *player* (the on-line algorithm) and an *adversary* (the off-line algorithm), whose goal is to produce a request sequence that would force the player to perform poorly. The off-line adversary assumed in the traditional competitive ratio model is a very strong and "malicious" one, and has been criticized by practitioners for not capturing the more typical situations. Although in some cases (e.g., routing problems, life-critical applications, etc.) no assumptions can/should be made about the distribution of the incoming requests, in others researchers have sought to model "weaker" adversaries in order to adapt to the problem at hand.

Of particular relevance are problems where some partial knowledge, or "look-ahead," of the requests is available in advance to the on-line algorithm. For the case of paging and caching, the effect of look-ahead has been analyzed in [1] and [21]. In the current paper, we show the improvement on the competitive ratios from linear to logarithmic when the algorithm is allowed to wait a (small) time before notifying the users the decision on their requests, and to a constant when the algorithm can delay notifications for up to a constant fraction of the movie length.

1.2 Contributions of this paper

The contributions of this paper are two-fold. First, we formulate the `VOD` problem from a resource optimization perspective, and give a characterization (upper and lower bounds) of the competitive ratio of movie scheduling algorithms as a function of the notification time. More specifically, let c denote the number of available channels, u the total number of potential users, T the typical duration of a movie, and ν the notification time (that is, the time the system is allowed to wait before notifying the users whether their requests have been accepted). Also let $\mu \stackrel{\text{def}}{=} u/c$. We measure the performance of movie-scheduling algorithms by the *revenue* they produce. (We can assume that the operating costs are fixed, so the profit is a function of the revenue only.) For simplicity, assume that each request that is accepted yields one unit of profit. Our results on the competitive ratio are summarized in Table 1.

COMPETITIVE RATIO	*Notification Time*		
	Immediate ($\nu = 0$)	*Short* ($\nu = O(T/\sqrt{\mu})$)	*Long* ($\nu = O(T)$)
Lower bound	$\frac{u}{c+1} \approx \mu$	$\Omega(\ln \mu)$	$\Omega(1)$
Algorithm	μ	$O(\ln \mu)$	$O(1)$

Table 1. *Competitive ratio of movie scheduling algorithms as a function of the notification time*

One important property of Adaptive `VOD` is the chance of a request being rejected even though there is available capacity to serve it. We call this property *refusal by choice*. The figures shown in Table 1 are for when refusal by choice applies. For completeness, we also consider the case when refusal by choice is prohibited, and show a lower and upper bound both equal to μ. This means that if the system cannot reject requests, then the (worst-case) performance of any Adaptive `VOD` algorithm cannot improve compared to the Full `VOD` approach.

For the case of refusal by choice with a small delayed notification, we show a lower bound of $\ln \frac{\mu}{2}$ on the competitive ratio of any algorithm, and introduce a class of algorithms (which we call `Pre-partition`) that, for a big enough number of channels, exhibit an asymptotically optimal behavior. In particular, the *Harmonic* partition version, which we analyze, achieves a competitive ratio of $1 + 8H_\mu$, where H_μ is the μth harmonic number.

Finally, for the case when the length of the movie and the notification time are linearly related (i.e., $\nu = \frac{T}{\ell}$, for some constant ℓ—for example, notifying users within 15 minutes after their requests for 2-hour movies), we present a simple algorithm (`SF`, for "Serve-a-Fraction") which achieves constant competitiveness (specifically, its competitive ratio is $\ell + 1$). We also show that, roughly, this is the best that any on-line algorithm can achieve.

Competitive analysis gives us an understanding of what we are up against in worst-case scenarios. How well would these algorithms perform in practice? Inspired by our results above (specifically, constant competitive ratio for long notification time), in this paper we also propose a specific design for `VOD` services—"*Shared* `VOD`"—and show by simulations that this kind of service may be a valuable complement to either Full `VOD` or Near `VOD`. Roughly, Shared `VOD` is a scheme with *fixed* start times on all channels, which accommodates (and benefits from) the most natural user behavior (particularly, the so-called "*lazy tune-in*" phenomenon: users with no strong preference are more likely to join movies that are already being scheduled). Essentially, we find that Near `VOD` is best suited for high-load, non-selective audience situations; Full `VOD` is best suited for low-load, highly selective audiences; and Shared `VOD` provides a flexible solution for the "in between" situations. Roughly, in order to generate the same total revenues as Shared `VOD`, a Full `VOD` service would have to charge $2-5$ times as much per movie, while the offerings of a Near `VOD` service would have to be acceptable to at least 20%-50% of the viewers.

Our analysis here is "typical-case," in the sense that we assume that some knowledge is available of the system's load. In this framework, we propose and evaluate a variety of on-line channel allocation algorithms: `Greedy`, `Harmonic`, and `Serve a Fraction` (`SF`), introduced in this paper. We compare these strategies to a near-optimal off-line strategy—namely, a strategy which "knows" the loading factor and distribution of requests in advance—and show, in particular, that the class of `SF` algorithms substantially outperforms the rest. It turns out that the truly optimal off-line algorithm is not amenable to implementation; fortunately, the variant we implemented is optimal up to a factor of 2, as we also show.

1.3 Related work

The assignment of users to resources, as embodied by the `VOD` problem, has a natural graph-theoretic formulation. This has been recently done in [2], which considers an on-line version of the *set cover* problem. For the case of what we call immediate notification they show that the linear bound can be improved to (super)logarithmic when randomization is allowed. They also generalize the problem to a setting in which every customer discloses a list of alternative movie titles he/she wants to watch (*non-disjoint max cover*).

In a recent development, in [5] it is shown that by exploiting the knowledge on the distribution of the requests for movies, there is a randomized strategy that also achieves a constant competitive ratio for the case of delayed notification mentioned above. Interestingly, the exact form of the distribution of requests need not be known in advance, as a version of their scheduling algorithm is able to follow (slow) changes in the distribution of requests.

Finally, the VOD problem is not unrelated to the more "classical" problems of (parallel) machine scheduling (e.g., [6,19]) and call control (e.g., [7,15]), to mention a few. However, while the emphasis in these problems is on several users being excluded from the concurrent use of a given resource, VOD has the distinct/opposite property of allowing the channel assignment algorithm to serve several users simultaneously on the same channel.

1.4 Organization of the paper

The remainder of the paper is organized as follows. Section 2 presents the model for VOD and the necessary definitions. Section 3 deals with the investigation of the problem using competitive analysis. In Section 4 we present the new VOD scheme we are proposing, together with simulation results. (Sections 3 and 4 are relatively self-contained, so the familiar reader with system design and evaluation goals can skip the first section and proceed directly to the second.) We conclude with some final remarks and directions for future work.

2 Model and Definitions

One of the motivations for this work is the problem of efficient sharing of a limited broadcast capability, controlled and supplied by a single server. This model is represented in Figure 3. This seems to be a reasonable model for the two likely bottlenecks for VOD identified in Section 1. The model considers finite, fixed sets of movies, users and channels, denoted respectively by M, U and C. Denote the cardinality of these sets by the corresponding lower case letter, i.e., m, u and c, respectively.

The system has three positive parameters related to time:

- T – the duration of a movie. (For the purpose of this paper, and without much loss of generality, we assume T is the same for all movies, and that no user can place more than one request in each interval of length T.)

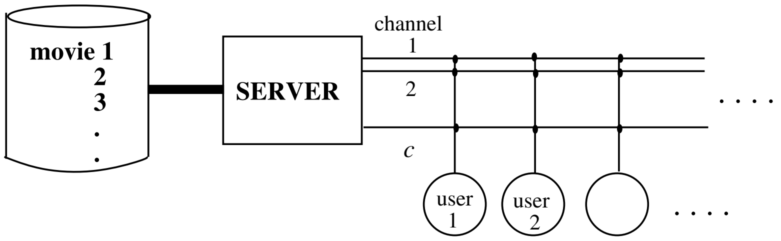

Fig. 3. *Model for Video on Demand*

- ν – the *notification time*, the maximal delay between a request and its response.
- τ – the *service time*, the maximal delay between a request and the start of the movie (if it is accepted).

We assume that $\nu \leq \tau < T$. A *request* is a tuple $(time, user, movie)$ where *time* is a positive number, $user \in U$ and $movie \in M$. Similarly, a *response* is a tuple (*time,user,movie,channel, servetime*) where $channel \in C$ and *servetime* is either real number $\in [time, time+\nu]$, or -1. If $servetime = -1$, we say that the response is rejected. A sequence of requests (t_i, u_i, m_i) is *valid* if the time values t_i are monotonously increasing and there is at least time T between any two subsequent requests of the same user.

Intuitively, the movie scheduling algorithm determines the responses at any moment, based on the requests up to that moment. More formally, a *scheduling algorithm* is a function from a time, a state, and a sequence of requests (all before the given time), to a time (for next wakeup, if no requests would occur before), a new state, and possibly one or more responses.

We consider different types of movie scheduling algorithms, based on the definitions above. An algorithm `A` is said to *refuse by choice* if in some run it has a response $(t_i, u_i, m_i, c_i, s_i)$ with $s_i = -1$, while there is some free channel at t_i. A channel c_k is *free* at time t if there is no response $(t_k, u_k, m_k, c_k, s_k)$ with $t_k \leq t$, $s_k > 0$, and for which $[s_k, s_k + T]$ intersects with $[t, t + T]$. An algorithm `A` is said to perform *immediate notification* if each request must be decided upon as soon as it is presented, i.e., $\nu = 0$. (Note that if a request is accepted, `A` must serve it within time τ.) Algorithms that do not always notify immediately are said to perform *delayed notification*, and in this case they have to notify within the parameter $\nu > 0$ defined above.

We remark that we consider the special case of *non-preemptive* video scheduling algorithms, and work exclusively with *flat* charging policies (i.e., the service provider charges a fixed rate for meeting a request, irrespective of the delay in meeting it), although our results can be generalized to hold for other charging policies. A detailed study of this subject of fair pricing is a topic for future research.

We now give a more technical definition of the competitive ratio [18]. A measure of performance of movie scheduling algorithms is the *revenue* they produce. For the purpose of this paper, a *revenue function* is a function that maps responses to the rationals (income). If $s_i = -1$, then $rev(t_i, u_i, m_i, c_i, s_i) = 0$. The revenue of a run of the system is the sum of the revenues of all the responses in the run. Let $\mathtt{A}(\rho)$ denote the run produced by algorithm $\mathtt{A}$ when presented with request sequence ρ (we assume that $\mathtt{A}$ is deterministic). The *competitive ratio* of on-line algorithm $\mathtt{A}$ is defined as

$$\kappa(\mathtt{A}) = \sup_{\mathtt{A}^*,\rho} \frac{rev(\mathtt{A}^*(\rho))}{rev(\mathtt{A}(\rho))}$$

where $\mathtt{A}^*$ is an off-line algorithm and ρ is a sequence of requests. As mentioned before, it is sometimes convenient for the analysis to describe things in terms of a game between a player (the on-line algorithm) and an *adversary* (the off-line algorithm), whose goal is to produce a request sequence that would force the player to perform poorly.

3 Competitive Analysis of Video on Demand

In this section we show upper and lower bounds on the competitive ratio $\kappa(\mathtt{A})$ of on-line movie scheduling algorithms $\mathtt{A}$, as a function of the notification time ν (see Table 1). Intuitively, there must exist algorithms with a competitive ratio of μ, because by "bunching up" requests, the adversary can serve at most u requests for every c requests the on-line algorithm serves. We first formalize this argument and show that there indeed exists a simple algorithm $\mathtt{A}_1$ with a competitive ratio of μ. This algorithm is essentially the same as the Full VOD approach.

We then show that any algorithm $\mathtt{A}$ which is *not allowed refusal by choice* must have a competitive ratio of at least μ. Thus, no online strategy can, in the worst case, "gain" anything by bunching up requests with respect to the simple strategy $\mathtt{A}_1$, unless it is allowed to reject requests.

Next, we show that even refusal by choice, by itself, is not enough to improve worst-case performance. The timing of *notification* also determines how well an algorithm can do. Specifically, we show that an algorithm that performs *immediate notification* (i.e., $\nu = 0$) must have a competitive ratio of at least $\frac{u}{c+1}$ (which is nearly equal to μ). Thus, such algorithms do not have any advantage, in the worst case, over those which do not refuse.

We then consider the case of algorithms which are allowed to refuse by choice together with *delayed notification*. As expected, they may perform substantially better than those with immediate notification. More specifically, for the case of small notification times, we show a lower bound of $\Omega(\ln \mu)$ on the performance of such algorithms, and present a simple algorithm (`Harmonic`) which matches this bound in the case $\nu = \tau$. We finally show that these bounds improve to a constant when notification times are allowed to be long enough. The algorithm we present there (`SF`) is the basis for the new `VOD` scheme we propose in Section 4.

3.1 No refusal

In this section we show that a competitive ratio of μ is easy to achieve by giving a "naive" algorithm that achieves this ratio (and which is not allowed to turn down requests). For simplicity, we consider the special case of the service time $\tau = 0$; the proof is easily extended to any arbitrary delay τ. Consider the following algorithm $\mathtt{A}_1$: Given request sequence $\rho = r_1, r_2, r_3, ..., r_n$, $\mathtt{A}_1$ assigns r_i to any channel that is free at time t_i; otherwise r_i is rejected.

Theorem 1 *Algorithm* $\mathtt{A}_1$ *is* μ*-competitive.*

Proof: Each request in ρ is either accepted or rejected. Let n_A and n_R be the number of accepted and rejected requests, respectively. Thus, $n = n_A + n_R$.

If request r_i is rejected, it is rejected because exactly c previously accepted requests are being served at that instant. For each of these c requests r', we say r_i *is rejected because of* r'. Define the "rejection graph" to be the undirected graph whose vertices consist of all n requests, and for any pair of vertices r_a and r_b, there is a link connecting them if and only if r_a is rejected because of r_b, or vice-versa. (Note that each rejected request must have exactly c incident links, and that the total number of links incident on all accepted requests must equal the total number of links incident on all rejected requests.)

Now we claim that each accepted request cannot have more than $u-c$ incident links. For suppose there exists an accepted request r_i which has $u - c + d$ incident links, i.e., $u - c + d$ ($d \geq 1$) requests (comprising, say, set R) are rejected because of r_i. Consider the latest request in set R, say, request r_l at time t_l. Note that $t_l < t_i + T$. Thus, there were exactly $u - c + d$ rejections in the time interval $[t_i, t_l]$. Further, since r_l was rejected, there were exactly c acceptances in the time interval $(t_l - T, t_l]$. It follows that there were at least $(u-c+d)+c = (u+d)$ requests in $(t_l - T, t_l]$, which contradicts the fact that each user of the u users may have at most one request during $[t_l - T, t_l]$.

Thus, the total number of links incident on accepted requests is at most $n_A(u - c)$, and the number incident on rejected requests is exactly $n_R c$. As a consequence, we have $n_A(u - c) \geq n_R c$, or $n_A u \geq nc$, which implies $\frac{n}{n_A} =$

$\kappa(\mathtt{A}_1) \leq \frac{u}{c}$. □

The following theorem shows that, under the circumstances, this is the best that can be done.

Theorem 2 *Let $T > \tau$. Any algorithm* $\mathtt{A}$ *that is not allowed to refuse by choice must have a competitive ratio $\kappa(\mathtt{A}) \geq \mu$.*

Proof: Let $\mathtt{A}$ be an algorithm that does not refuse by choice. We construct request sequences ρ that would necessitate a competitive ratio of at least $\mu = u/c$.

Let ρ be a request sequence for $\mathtt{A}$, which begins with a set of c requests for c different movies, presented to $\mathtt{A}$ at time 0. Assume there is no additional request until time ν. Algorithm $\mathtt{A}$ must schedule all of the first c requests until time ν, since it does not refuse by choice. Furthermore, the schedules should all start before time τ.

We now have two mutually exclusive cases:

(1) Suppose that at least one movie is scheduled before τ; let m_i be such a movie. Then ρ would contain $u - c$ requests for the same movie m_i, all arriving exactly at time τ. However, at this time all channels are already scheduled, and since we do not allow preemption, the online algorithm would have to refuse all of these movies.

Thus, $\mathtt{A}$ honors a total of c requests in its run. On the other hand, obviously there exists an offline algorithm $\mathtt{A}^*$ which can serve all the requests. Thus, $\kappa(\mathtt{A}) = u/c$.

(2) Suppose that $\mathtt{A}$ schedules each of the c movies to begin at time τ. The request sequence ρ would continue with $u - c$ requests at time $T - \frac{\tau}{2}$ for some movie. Of course, algorithm $\mathtt{A}$ cannot honor any of these requests since all channels are still occupied at that time. Thus, $\mathtt{A}$ honors a total of c requests in this run.

On the other hand, there exists an offline algorithm $\mathtt{A}^*$ that honors all requests by honoring the first set immediately and the second set at time T (within the permitted delay). Thus, again, $\kappa(\mathtt{A}) = u/c$. □

3.2 Immediate notification

One might wonder whether an algorithm that notifies immediately could perform substantially better if allowed to reject requests. As the following theorem shows, that is not the case: Refusal by choice, by itself, is not enough to improve worst-case performance. The timing of notification also determines how well an algorithm can do.

Theorem 3 *Any algorithm* A *capable of refusal by choice, but which is required to notify immediately, must have a competitive ratio* $\kappa(\mathtt{A}) \geq \frac{u}{c+1}$.

Proof: Let A be an algorithm that is allowed to refuse requests by choice, but which always notifies immediately. Let it have a competitive ratio $\kappa(\mathtt{A}) = x$. The proof is by contradiction. It is based on the fact that if A has a lower competitive ratio, the adversary can construct a request sequence that makes A violate the ratio. The adversary first presents a series of "baits" such that A is forced to fill up its channels in order to maintain its competitive ratio. Once A is filled up, the adversary then presents as many requests as possible; thus, the adversary can take all of these while A can take none.

Specifically, the adversary first presents a request sequence $\rho' = r_1, r_2, ...r_l$. Each request r_i is a request for movie m_i made at time $i \cdot \epsilon$, where ϵ is some arbitrarily small interval of time. The adversary constructs the request sequence according to the following rules:

(1) If r_i is rejected by A, $m_{i+1} = m_i$;
(2) if r_i is accepted by A, $m_{i+1} \notin \{m_1, m_2, ...m_i\}$; and
(3) the total number of different movies requested in ρ' is c.

Thus, the adversary presents individual requests for movie m (at instants separated by ϵ) until one of these is accepted by A, then it presents a group of individual requests for another movie m' until one is accepted, and so on, for a total of c request groups for c different movies. The total number of requests accepted by A is c (the last one in each of the c groups).

On the other hand, by "bunching up" the requests in each group and transmitting a single movie to serve an entire group, the adversary can accept each one of these l requests. Since A must maintain a competitive ratio of x, A's ith acceptance must occur by request r_{ix}. Thus, the total number of requests is $l \leq cx$. Note that since there are c groups of requests of total size $l \leq cx$, there must exist a group G of size $g \leq x$.

Having "filled up" all the channels of A, the adversary now presents a request sequence $\rho'' = r_{l+1}, r_{l+2}, ...r_u$ for any one of the movies not requested so far. It accepts all the requests from the earlier sequence ρ', except for the $g \leq x$ requests in group G. Thus, at this point the adversary has one channel free; it accepts all the requests in ρ'' and serves them on this channel.

The online algorithm A has served exactly c requests, while the adversary has served $(l-g)$ requests from ρ' and $(u-l)$ from ρ'', for a total of $(u-g) \geq (u-x)$. Thus, to preserve a competitive ratio of κ, a necessary condition is $x \geq \frac{u-x}{c}$, which reduces to $x \geq \frac{u}{c+1}$. □

Remark 4 *The above lower bound can be readily extended to non-zero (but* *small) values of ν, namely, for all $\nu < \frac{\tau}{c\kappa}$.*

3.3 Short notification yimes and pre-partition algorithms

In this section we give bounds on the competitive ratio of movie scheduling algorithms that are allowed refusal by choice and short notification times. For the case of the upper bound we also assume big, but yet reasonable, values of c, the number of channels (for example, $c \approx 65$ for $\mu = 20$). We first show a lower bound of $\ln(\mu/2)$ on the competitive ratio of such algorithms for the case of short notification times (specifically, $\nu < \frac{2\tau}{\mu}$). This result is somewhat technical, and resembles (in spirit) the lower bound reasoning of the previous section. The reader more interested in "positive" results is advised to proceed to the subsequent section, Pre-partition algorithms.

Let $\mathtt{A}$ be an online algorithm. We prove the lower bound result using a particular type of game between $\mathtt{A}$ and its adversary $\mathtt{A}^*$. The structure of the game will be as follows. The adversary will present $\mathtt{A}$ with several sets $R_1, R_2, ...$ of requests in quick succession. When presented with each set of requests, $\mathtt{A}$ has to schedule just enough to maintain a competitive ratio of x. However, the adversary chooses the request sets in such a manner that it is able to force $\mathtt{A}$ to accept "cheap" requests, and thereby block its channels to further, more remunerative requests, which are served by the adversary. If the adversary $\mathtt{A}^*$ ends up serving at least x times more requests than $\mathtt{A}$, we will have shown that $\mathtt{A}$ cannot have a competitive ratio of x. This allows us to state:

Theorem 5 *Any algorithm $\mathtt{A}$ capable of refusal by choice with notification time $\nu < \frac{2\tau}{\mu}$ must have a competitive ratio $\kappa(\mathtt{A}) > \ln \frac{\mu}{2}$.*

We first describe the game in more detail, and establish some technical facts. Suppose there is an x-competitive online movie scheduler algorithm $\mathtt{A}$. Consider the following sequence of requests. At time 0, request set R_1 is presented, which contains one request for each of c movies. $\mathtt{A}$ must eventually serve at least $\frac{c}{x}$ of them (in fact, $\lceil \frac{c}{x} \rceil$) to maintain a competitive ratio of x, otherwise $\mathtt{A}^*$ could serve all of them and have a ratio greater than x. We say that $\mathtt{A}$ has *committed* to at least $\frac{c}{x}$ requests. Note that $\mathtt{A}$ must do so by time ν.

At time ν, $\mathtt{A}$ is presented with a set of requests R_2, which is composed of existing requests from R_1 and some additional requests as follows: for each of the $c - \frac{c}{x}$ movies not committed in the first decision there is one additional request, and two requests for $\frac{c}{x}$ new movies. Overall, the set R_2 contains two requests for each of c movies. Again, $\mathtt{A}$ must serve at least $\frac{c}{x} \cdot \frac{1}{2}$ movies (each with 2 requests), otherwise $\mathtt{A}^*$ could satisfy all and accrue $2c$. In other words, the adversary would have gotten revenue of $2c$ while $\mathtt{A}$'s revenue would be $\frac{c}{x}$ from the movies taken by time ν plus *less than* $\frac{c}{x}$ from the additional movies taken by time 2ν (two movies per channel, but less than $\frac{c}{2x}$ channels).

This generalizes to request set R_i at time $(i-1)\nu$ in the following way: At time $(i-1)\nu$ there are i requests for each of c movies which are not yet being served. Again, old outstanding requests are reused. After set R_{i-1} at time $(i-2)\nu$ is presented, `A` has to serve $\frac{c}{(i-1)x}$ movies, each with $i-1$ requests. There are $c - \frac{c}{(i-1)x}$ movies left, with $i-1$ outstanding requests; the adversary will just add one additional request for each one of these movies. For the remaining $\frac{c}{(i-1)x}$ channels, the adversary will produce i requests for new movies. Thus, the total number of new requests is $(c - \frac{c}{(i-1)x}) + i \cdot \frac{c}{(i-1)x} = c + \frac{c}{x}$. Again, `A` must satisfy at least $\frac{c}{x} \cdot \frac{1}{i}$ of these requests, otherwise the adversary could satisfy them all and beat the competitive ratio. If this was the case, the adversary would have gotten revenue of $i \cdot c$, while `A`'s revenues would be $\frac{(i-1)c}{x}$ from the movies taken before time $(i-1)\nu$, plus *less than* $\frac{c}{x}$ from the movies taken from R_i at time $(i-1)\nu$ (i movies per channel but less than $\frac{c}{i \cdot x}$ channels). Thus, the adversary would achieve a ratio greater than x. This process continues until `A` exhausts its supply of channels; assume this happens after R_k is presented. At this point, the adversary makes all remaining users request the same movie. Unlike `A`, the adversary will be able to serve all these requests, as explained later.

In the game described above, we can assume that `A` knows `A`*'s strategy, and therefore for a given competitive ratio x, `A` should commit to as few movies as possible from sets $R_1, R_2, \ldots$ Otherwise, `A` would serve less users with the commited movies, which would leave the adversary with more users at the end of the process.

Consequently, we get the following condition on k: $\sum_{i=1}^{k} \frac{c}{i \cdot x} \geq c$, which is equivalent to

$$H_k = \sum_{i=1}^{k} \frac{1}{i} \geq x \ , \tag{1}$$

where H_k is the kth harmonic number.

We also need to make sure for our argument to hold that the number of requests is not too large to exhaust the supply of users in a period of length T. The size of the first request set is c, while the size of each of the $k-1$ subsequent sets is $(c + \frac{c}{x})$. Thus, the total number of requests needed by this strategy is $c + (k-1)(c + \frac{c}{x})$, which must be less than u. This imposes the following condition on k:

$$k + \frac{k-1}{x} \leq \mu \ . \tag{2}$$

In order for the adversary to take advantage of the situation by accepting requests that `A` cannot, we also require that the acceptance/rejection notifications for the kth set of requests be issued *before* time τ, to prevent `A` from scheduling the last, very profitable movie. This will require

$$k\nu < \tau \ . \tag{3}$$

Just after set R_k is presented, there exist kc unsatisfied requests (k requests for each of c movies). `A` has already committed to a total of $\frac{(k-1)c}{x}$ requests. `A` serves $\frac{c}{x}$ more requests at the kth round, for a total of $\frac{kc}{x}$ requests. On the other hand, `A`* serves all the kc unfulfilled requests. In addition, it presents set R_{k+1} at time $k\nu$, which contains as many requests as possible (to make a total of u requests), all for one of the movies that are not been served by `A`. At this point, the only requests not served by `A`* are the $\frac{(k-1)c}{x}$ requests that `A` committed to. Thus, the adversary is able to serve a total of $u - \frac{(k-1)c}{x}$ requests.

In order for algorithm `A` to preserve a competitive ratio of x, we must have

$$x \geq \frac{u - (k-1)\frac{c}{x}}{\frac{kc}{x}} = \frac{ux - (k-1)c}{kc} = \frac{x}{k}\mu - (1 - \frac{1}{k}) \ ,$$

which implies $(1 - \frac{1}{k}) \geq x(\frac{\mu}{k} - 1)$, or

$$x \leq \frac{1 - 1/k}{\mu/k - 1} \ . \tag{4}$$

We now proceed to prove the bounds on the competitive ratio.

Proof: Suppose there exists an online algorithm `A` with competitive ratio $\kappa(\mathtt{A}) = x \leq \ln(\mu/2)$. The adversary chooses $k = \lfloor \mu/2 \rfloor$. Since $\forall k$, $\ln(k+1) \leq H_k$, we have $\ln\lceil \mu/2 \rceil \leq H_k$. Thus, $x \leq H_k$, and constraint (1) is satisfied.

Regarding constraint (2), since $x \geq 1$ (by definition), we have $(k + \frac{k-1}{x}) \leq 2k$, always. If $k = \lfloor \mu/2 \rfloor$, we obtain $2k \leq \mu$, so constraint (2) is also satisfied. Thus the "game" can be completed. Finally, the assumption (stated earlier) that $\nu < 2\tau/\mu$ allows constraint (3) to be satisfied. We now check whether (4) is satisfied. The RHS of (4) is now

$$\frac{1 - \frac{1}{\lfloor \mu/2 \rfloor}}{\frac{\mu}{\lfloor \mu/2 \rfloor} - 1} \leq \frac{1 - \frac{1}{\mu/2}}{\frac{\mu}{\mu/2} - 1} = 1 - \frac{2}{\mu} \ .$$

Thus, the RHS is always smaller than 1. Since the LHS of (4), x, is larger than

1, (4) fails to hold, and therefore no algorithm can have a competitive ratio of $x \leq \ln(\mu/2)$. $\square$

Remark 6 *An $\Omega(\ln \mu)$ (with a smaller constant factor) can also be achieved even if the adversary only serves the last request. In this case, it is only needed that the last set of requests be issued before time T, enough to prevent* `A` *from scheduling the last movie. This way, the lower bound can be extended to hold for all $\nu \leq \tau$. Furthermore, the bound can also be readily strengthened to μ for small values of c. (Think of $c = 1$, where* `A` *eventually accepts one request, and then all the remaining users request the same movie.)*

3.3.1 Pre-partition algorithms

In this section we present algorithm `Harmonic` which, when allowed short notification time and refusal by choice, and given a large enough (but reasonable) number of channels, matches (asymptotically) the lower bound of the previous section. More specifically, $\kappa(\texttt{Harmonic}) \leq 1 + 8H_\mu$, where H_μ is the μth harmonic number. This algorithm has been recently formally modelled and its performance studied through discrete simulation in [16].

Roughly, `Harmonic` divides the c channels into μ sets $S_1, S_2, ...S_\mu$. Each set S_i contains $\frac{c}{iH_\mu}$ channels (in fact, $|S_i| = \lfloor \frac{c}{iH_\mu} \rfloor$). Thus, the total cardinality of the sets is $\frac{c}{H_\mu} \cdot (1 + 1/2 + ... + 1/\mu) = c$. The algorithm stipulates that any transmission made on a channel in set S_i must serve at least i requests (see Figure 4).

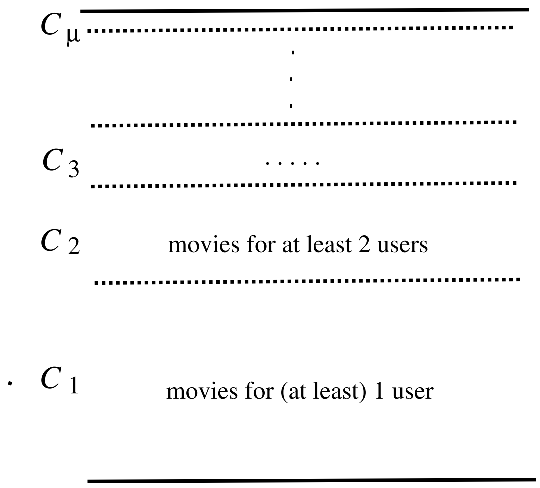

Fig. 4. *The* Harmonic Prepartition *algorithm*

`Harmonic` schedules the arriving requests as follows: Each movie m_j is associated with a distinct *queue* Q_j of requests for that movie. Initially, $Q_j = nil$ for all j. When a request for m_j is made, it is immediately inserted into Q_j. The purpose of Q_j is to accumulate as many requests as possible for movie m_j, so that it can serve all of them with a single transmission at the end. Each queue Q_j is associated with a "start time" $start(Q_j)$, representing the time at which the first request in the queue was made.

At time $start(Q_j) + \nu$, the algorithm decides whether to serve the requests in Q_j (say k in number). If there exists a free channel in a set S_i with index $i \leq k$, it immediately notifies and begins a transmission for m_j on one such channel, thus serving all k requests, and Q_j is re-initialized to *nil.* (The performance of `Harmonic` stated below holds for an arbitrary choice of a free channel. In practice, however, one would choose a channel according to First-Fit, i.e., from the set with the smallest index.) If, on the other hand, there exists no such channel, `Harmonic` rejects only those requests in Q_j made at time $start(Q_j)$, deletes them from Q_j, and resets $start(Q_j)$ to the time of the earliest request now in Q_j. We are able to state the following.

Theorem 7 *Let $\nu = \tau$ and $c \geq (\mu + 1)H_\mu$, where H_μ is the μth harmonic number. Then κ(`Harmonic`) $= O(H_\mu)$.*

Proof: We define the *saturation level* at instant t to be the highest i such that *all* channels in sets $S_1, S_2, ...S_i$ are occupied at time t. The saturation level of an *interval* of time is the *highest* saturation level achieved during the interval.

Consider any execution of `Harmonic` in response to a request sequence ρ, and a corresponding execution of an offline algorithm $\mathtt{A}^*$ in response to the same request sequence. We divide the entire execution into *intervals* $I_0, I_1, ...I_l$ of length T. Interval I_j spans the half-open time interval $[jT, (j+1)T\ )$.

For all $j = 0, 1, ...l$, let $A(j)$ be the number of requests accepted by `Harmonic` in interval I_j, and $R(j)$ the number of requests that were *rejected by* `Harmonic` *in I_j but accepted by* $\mathtt{A}^*$ *in its execution.* Let σ_j be the saturation level of interval I_j. Thus, there exists some instant $t_j \in I_j$ in which all channel sets $S_1, S_2, ...S_{\sigma_j}$ are "full." If set S_i is full, then `Harmonic` serves at least $i \cdot |S_i| \geq i \cdot \lfloor \frac{c}{iH_\mu} \rfloor \geq \frac{c}{H_\mu} - i$ users. Thus, for a saturation level σ_j, `Harmonic` serves at least $\frac{\sigma_j c}{H_\mu} - \frac{\sigma_j(\sigma_j+1)}{2}$ users. Since these requests must have been accepted either in interval I_j or interval I_{j-1}, we have, $\forall j = 1, 2, ..., l$,

$$A(j-1) + A(j) \geq \frac{\sigma_j c}{H_\mu} - \frac{\sigma_j(\sigma_j+1)}{2} \geq \frac{\sigma_j c}{2H_\mu}\ , \tag{5}$$

given our assumption on c. Along similar lines, $A(0) \geq \frac{\sigma_0 c}{2H_\mu}$.

Now consider $R(j)$. By definition, all requests in $R(j)$ were rejected by `Harmonic` during I_j, so each such request was *made* in the interval $[jT - \nu, (j+1)T - \nu)$. The adversary $\mathtt{A}^*$ could thus serve these requests anytime during the interval

$[jT-\nu,(j+1)T)$. Hence the offline algorithm can utilize upto $2c$ transmissions to serve these requests (≤ 2 on each channel).

Assume first $\sigma_j < \mu$. Each such transmission by $\mathtt{A}^*$ must contain $\leq \sigma_j$ requests from $R(j)$. For if there existed an offline transmission serving more than σ_j requests from $R(j)$, `Harmonic` too would be in a position to accept these requests, since its saturation level in that interval was σ_j. Hence we have, $\forall j$,

$$R(j) \leq 2\sigma_j c \ . \tag{6}$$

From (5) and (6) above, we have, $\forall j$,

$$4H_\mu(A(j-1)+A(j)) \geq R(j) \ . \tag{7}$$

The case $\sigma_j = \mu$ is handled similarly, yielding the same result. (In this case, however, the number of requests rejected by `Harmonic` but accepted by $\mathtt{A}^*$ is bounded by u, and the fact that no user can place two requests in one interval of length T; i.e., the argument here is that there cannot be that many requests.)

Let now A be the total number of requests accepted by `Harmonic`, and let R be the total number of requests rejected by `Harmonic` but accepted by $\mathtt{A}^*$. Note that $A = A(0)+A(1)+...+A(l)$, and similarly, $R = R(0)+...+R(l)$. Summing up the $(l+1)$ inequalities of the form (7) (and making the necessary adjustments), we obtain $4H_\mu \cdot (2A) \geq R$, or

$$R \leq 8H_\mu A \ . \tag{8}$$

From the definitions, $rev(\mathtt{A}^*) \leq A+R$, while $rev(\mathtt{Harmonic}) = A$. Hence, from (8), $\kappa(\mathtt{Harmonic}) \leq 1+\frac{R}{A} \leq 1+8H_\mu$. □

3.4 Long notification times and the Serve-a-Fraction algorithm

In this section we present a simple algorithm, `SF` (for "Serve-a-Fraction") which, when allowed a long-enough notification time, achieves constant competitiveness. We also show that its performance is optimal. Specifically, we assume in this section that the notification time and the duration of the movies are linearly related; that is, $\nu = \frac{T}{\ell}$, for some constant ℓ. (For example, a typical value of T could be 2 hours, while ν 30 minutes, yielding $\ell = 4$.)

Algorithm `SF` proceeds as follows. It divides the time into fixed time intervals of length ν. (Thus, ℓ also represents the length of the movie measured in number of intervals.) Requests are collected during each interval, at the end of which a decision is made on which requests to serve. These requests are served immediately. As its name indicates, only a fraction of the channels are utilized at each interval; this fraction equals $\frac{c}{\ell}$. (We assume, for simplicity of exposition, that c is a multiple of ℓ.) The requests that are chosen to be served in those channels are the most popular requests in that interval. Note that, from the definitions, there are always enough channels available at each interval.

Thus, the execution of the algorithm can be divided into intervals I_i, $1 \leq i \leq n$, of length ν. For $1 \leq i \leq n$ and $1 \leq j \leq m$, let $w(i,j)$, $w : \{1,\ldots,n\} \times \{1,\ldots,m\} \to \{1,\ldots,u\}$, denote the number of viewers who request the j-th movie in the ith interval. We refer to $w(i,j)$ as the "weight" of movie j in I_i. We assume that movies are sorted by their weight. (I.e., $w(i,j)$ is a non-increasing function for any fixed i and thus if $j' > j > 0$ then $w_i(j') \leq w_i(j)$ for any $1 \leq i \leq n$.) We let $f \triangleq \frac{c}{\ell}$ denote the fraction of channels that `SF` will be serving.

Theorem 8 $\kappa(\mathtt{SF}) \leq 1 + \frac{T}{\nu}$. *(Note that due to the long notification time assumption, this is a constant.)*

Proof: Consider any execution of `SF` in response to a request sequence ρ, and a corresponding execution of an off-line algorithm $\mathtt{A}^*$ in response to the same request sequence. For all $i = 1, ...n$, let $A(i)$ be the number of requests accepted by `SF` in interval I_i, and $R(i)$ the number of requests that were *rejected by* `SF` *in* I_i *but accepted by* $\mathtt{A}^*$ *in its execution.*

Consider interval I_j, and assume that the number of different movies accepted by `SF` during that interval is f_j Let $w_1^j, w_2^j, \ldots, w_{f_j}^j$ denote the weight of the f_j most popular movies. Then, by definition of `SF`, we have, $\forall j = 1, 2, ..., n$,

$$A(j) \geq \frac{c}{\ell} \cdot w_{f_j}^j \ . \tag{9}$$

Now consider $R(j)$. By definition, all requests in $R(j)$ were rejected by `SF` during I_j, so each such request was *made* in the interval $I_{j-1} \cup I_j$. The adversary $\mathtt{A}^*$ could thus serve these requests during the interval $I_{j-1} \cup I_j \cup I_{j+1}$. Hence, $\mathtt{A}^*$ can utilize up to c transmissions to serve these requests, provided $\frac{T}{\nu} = \ell \geq 3$. Each such transmission by $\mathtt{A}^*$ must contain $\leq w_f^j$ requests from $R(j)$, for if there existed an off-line transmission serving more than w_f requests from $R(j)$, `SF` too would have accepted these requests. Hence we have $\forall j$,

$$R(j) \leq c w_f^j \ . \tag{10}$$

Let now A be the total number of requests accepted by `SF`, and R be the total number of requests rejected by SF but accepted by $\mathtt{A}^*$. Thus, $A = A(1) + A(2) + \cdots + A(n) \geq \frac{c}{\ell}\sum_{j=1}^{n} w^j_{f_j}$, and similarly, $R = R(1) + \cdots + R(n) \leq c\sum_{j=1}^{n} w^j_f$. From the definitions, the revenue of $\mathtt{A}^*$ $rev(\mathtt{A}^*)$ obeys $\leq A + R$, while $rev(\mathtt{SF}) = A$. Hence, $\kappa(\mathtt{SF}) \leq 1 + \frac{R}{A} \leq 1 + \ell$. □

Remark: When $\ell < 3$, $\kappa(\mathtt{SF})$ deteriorates to $1 + 2\ell$, as the adversary can now use $2c$ transmissions in the reasoning above to serve the requests rejected by SF in a given interval.

It turns out that SF's competitiveness is nearly optimal, as the next theorem states.

Theorem 9 *Let $\frac{T}{\nu} = \ell$. Then $\kappa(\mathtt{A}) \geq \ell$, for any on-line algorithm* A.

The proof of the theorem is similar in spirit to that of Theorem 5, and is left as an exercise to the reader. In conclusion, what Theorem 8 says is that, under every possible distribution of requests, SF always accrues at least a constant fraction of the revenue of the ideal algorithm which would know all the requests beforehand, and could optimize thereof. Furthermore, this happens in a setting where the users are made to wait for the outcome of their requests a seemingly acceptable time. This leads us to the proposal of a concrete VOD service, the subject of the next section.

4 Sharing Video on Demand

In this section we present the new VOD scheme we are proposing. Effectively, the proposal is a combination of Full VOD with a Shared VOD scheme. (The service can be extended to accommodate a Near VOD option as well.) The general idea is as follows. The user is initially presented with the alternatives of having a dedicated, fully-functional service (i.e., FF, REW, PAUSE, etc.), or a shared, cheaper service. Should the user choose the latter option, the system then displays the movies that have already been requested by other users. The user may then choose to add his/her request for one of the existing movies, or place a new request. (Intuitively, a movie with more "weight" might have a better chance to be selected by a channel assignment algorithm. Also, this "advertising" of the movies already requested tries to capitalize on the "lazy tune-in" phenomenon—users with no strong preference are more likely to join movies that are already being scheduled—and might lead to more efficient resource utilization.)

Our proposed implementation of the Shared VOD part of the service consists of fixed start times (e.g., 8:30pm; 9:00pm; etc.) for the movies. There is a *notification time* for each interval I (e.g., 5 min. before the start time), at

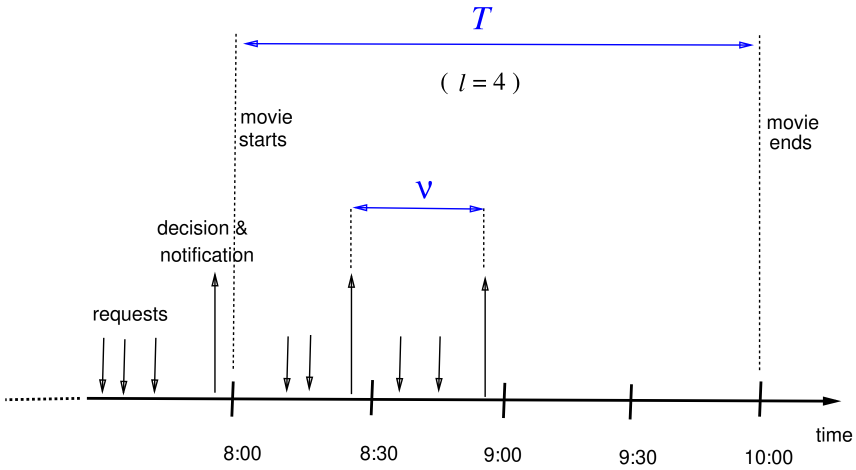

Fig. 5. *Implementation of Shared* `VOD`

which the channel assignment algorithm, presented with the sets of requests collected during that interval, performs a selection of the requests to be shown, and notifies the users. Our proposed implementation of Shared `VOD` is depicted in Figure 5.

We will be performing a typical-case analysis here, in the sense that we will assume that some knowledge is available of the system's load. In this framework, we will evaluate a variety of on-line channel allocation algorithms: `Greedy`, `Harmonic`, and `SF`, introduced in this paper. We will compare these strategies to a near-optimal off-line strategy—namely, a strategy which "knows" the loading factor and distribution of requests in advance. The off-line algorithm is based on a dynamic programming technique, which might be of independent interest. Our evaluation shows, in particular, that the `SF` algorithm substantially outperforms the rest.

4.1 On-line channel allocation strategies

Before we turn to the evaluation of the on-line channel allocation strategies mentioned above, we first need some preliminary definitions. The output of any algorithm that solves the movie scheduling problem is a list of movies that are to be shown. For each movie the interval and the channel to which it is assigned is specified as well. A solution to the problem is *feasible* if at any interval I_i, $1 \leq i \leq n$, the number of movies scheduled in intervals $I_{i-\ell+1}, \ldots, I_i$ is no greater than c (the number of channels). If $i-\ell+1 < 1$ then the condition is that the number of movies scheduled in intervals $I_1, \ldots, I_i$ is no greater than c. A solution is *optimal* if, given our revenue assumptions, the overall number of viewers of all movies scheduled is maximal among all feasible solutions.

The algorithms we present in this section are on-line, in the sense that they are to determine the responses at any moment based on the requests received up to that moment. That is, only the values of $w(i', j)$, $1 \leq i' \leq i$, are known to the algorithm at the notification time of the ith interval. We use f_i to denote the number of free channels at I_i.

For our evaluation, we consider the following channel allocation strategies (for convenience, we repeat the description of algorithms `Harmonic` and `SF` here):

`Greedy` At interval I_i, $1 \leq i \leq n$, schedule the first f_i movies (i.e., the f_i movies with highest weight).

`Harmonic` Divide the c channels into μ sets $C_1, C_2, ...C_\mu$. Each set C_i contains c/iH_μ channels, where H_μ is the μth harmonic number. The algorithm stipulates that at any interval, any movie shown on a channel in set C_k, $1 \leq k \leq \mu$, must have weight at least k (namely, it must have been requested by at least k users). See Figure 4.

`Serve-a-Fraction (SF)` At each interval I_i, schedule the first (i.e., highest-weight) $\frac{c}{\ell}$ movies, where ℓ is the length of the movie (measured in number of intervals).

Although `Greedy` might perform well in many typical scenarios, it is easy to see that $\kappa(\texttt{Greedy}) = \mu$. The particular input instance that makes `Greedy` perform so poorly is one in which c requests for c different movies occur in one interval, and in the next many (say, $\approx u$) requests ask for one (the same) movie. Since u is expected to be order(s) of magnitude larger than c, the potential for this type of behavior is something we would like to avoid.

In Section 4.3 we will be evaluating the performance of the channel allocation strategies mentioned above under typical distributions of requests and loading factor. Moreover, we will compare how well they do with respect to an optimal algorithm, i.e., one which has all the information to decide beforehand. We have developed such an optimal off-line algorithm to facilitate this comparison. In addition to helping in the evaluation of the on-line algorithms, this off-line approach may prove valuable elsewhere, in the design of algorithms which utilize some statistical predictions about the requests. We now turn to a sketch of its design.

4.2 Off-line algorithms

Given a distribution of requests over n time intervals, an obvious way of computing their optimal allocation is by exhaustively checking all possible combinations; this, however, is inefficient (i.e., it requires exponential computation). In this section we consider efficient (polynomial-time) algorithms.

Since off-line algorithms are given all the requests in advance, they can notify rejections at the beginning of the allocation process. Therefore, the notification time and service time are redundant parameters for off-line algorithms. The problem can now be abstracted and formulated as follows: Let $\mathcal{I}$ be a set of intervals. Let $w(I) \in \Re^+$ be a weight associated with each interval $I \in \mathcal{I}$. Define $w(J) = \sum_{I \in J} w(I)$ to be the weight of a set of intervals $J \subseteq \mathcal{I}$. We say a set $J \subseteq \mathcal{I}$ is *independent* if all the intervals in J are pairwise non-overlapping.

Given a positive integer c, the goal now is to find c disjoint independent sets $J_1, \ldots, J_c$ such that $\sum_{i=1}^{c} w(J_i)$ is maximum among all possible c disjoint sets of independent sets. Polynomial-time algorithms for the above-stated problem have been presented in [4,8]. These solutions, however, are based on a linear programming formulation of the problem and are not feasible to implement. In contrast, we have developed an algorithm which is more amenable to implementation, at the cost of being near-optimal. Specifically, its worst-case performance is at least half of that of the optimal algorithm.

We now describe our near-optimal off-line algorithm assuming that (i) the time axis is divided into n intervals, (ii) all movies start at the beginning of intervals, and (iii) all the movies are of length ℓ intervals. We will use the notations of the previous section. For the case $c = 1$ our near-optimal algorithm is optimal and is based on a dynamic programming technique. We first describe this optimal solution for one channel. We consider two cases:

$1 \leq i \leq \ell$ A solution to one channel in which there are $1 \leq i \leq \ell$ intervals can schedule at most one movie. Therefore, any optimal algorithm for $1 \leq i \leq \ell$ intervals schedules the movie with the highest weight over all the intervals, if such a movie exists.

$\ell < i \leq n$ Let A be the maximal possible number of viewers if we schedule a movie in I_i, the ith interval, and let B be the maximal possible number of viewers in an optimal solution for $i-1$ intervals. The latter is also the maximal possible number of viewers in an optimal solution for i intervals if such a solution does not schedule a movie in the ith interval. Note that if a movie is scheduled in I_i, no movie can be scheduled in intervals $I_{i-\ell+1}, \ldots, I_{i-1}$, and for the rest of the intervals the best is to take the optimal solution for $i - \ell$ intervals. Thus, the algorithm compares A with B. If $A \leq B$, then no movie is taken in I_i and the schedule remains the same as for $i-1$ intervals. Otherwise, the algorithm takes the best movie at I_i, "preempts" the movie scheduled for $I_{i-\ell+1}, \ldots, I_{i-1}$, if such movie exists, and the rest remains the same as in the $i - \ell$ schedule.

Our near-optimal off-line algorithm is a generalization of the above dynamic programming scheme. We derive the solution for c channels from the solution for $c - 1$ channels, where the solution for one channel described above serves as the base case. Given a solution for $c - 1$ channels, the step procedure that creates a solution for c channels is very similar to the one-channel solution presented above. We call this algorithm `Max-IS`, since in each step it essentially finds a maximum weighted independent set in the interval graph induced by the requests. Let `OPT` denote the optimal off-line algorithm. We now show that the number of viewers served by `OPT` is at most twice the number served by `Max-IS`.

Theorem 10 $rev(\texttt{Max-IS}) \geq \frac{rev(\texttt{OPT})}{2}$.

Proof: Denote by $m^i_1, \ldots, m^i_{k_i}$ the movies scheduled by algorithm `Max-IS` on channel i over a run of the algorithm, and let X_i be the total number of viewers who requested these movies. Let $X = X_1 + \cdots + X_c$. Let $\texttt{OPT} = Opt_0$ be the optimal solution and let W be the number of viewers served by this solution for all the movies in all the channels. We will transform Opt_0 into $Opt_1, \ldots, Opt_c$ in a sequence of steps. Let W_i be the number of viewers in solution Opt_i. We will then prove that

$$W_i \geq \max\{X_1 + \cdots + X_i \ ; \ W_0 - X_1 - \cdots - X_i\}$$

for all $1 \leq i \leq c$. Since $\texttt{Max-IS} = Opt_c$ and since $W = W_0$, it will follow that $X \geq W - X$ for $i = c$. This will imply that $X \geq \frac{W}{2}$.

We now describe how to generate Opt_1 from Opt_0; generating Opt_i from Opt_{i-1} will be done similarly. We proceed as follows. We add to channel 1 the movies $m^1_1, \ldots, m^1_{k_1}$ in the order they have been scheduled. Each time we add movies $m^1_i, \ldots, m^1_j$ we might have to delete movies $o_1, \ldots, o_j$ from solution Opt_0. Note, however, that the total number of viewers in the deleted movies is no greater than the total number of viewers in movies $m^1_i, \ldots, m^1_j$.

Assume by induction that we have completed the process up to the ith movie. Let $j > i$ be the minimal such that movie m_j is scheduled on another channel, say channel 2. Denote the movies scheduled on channel 1 by $B_1, m^1_i, o_1, o_2, \ldots, o_a, E_1$, where B_1 is the set of all the movies that appear before m^1_i (that is, $m^1_1, \ldots, m^1_{i-1}$), o_a is the first movie that overlaps in time with m^1_j, and E_1 is the set of the rest of the movies scheduled on channel 1. Similarly, denote the movies scheduled on channel 2 by $B_2, o'_b, o'_{b-1}, \ldots, o'_1, m^1_j, E_2$, where E_2 is the set of all the movies that appear after m^1_j, o'_b is the first movie from the end that overlaps in time with m^1_j, and B_2 is the set of the rest of the movies scheduled on channel 2.

Opt_1 now consists of the following. In channel 1 we schedule the movies $B_1, m^1_i, m^1_{i+1}, \ldots, m^1_j, E2$, while in channel 2 we schedule the best between $B_2, o'_b, \ldots, o'_1, E_1$ and $B_2, o_1, \ldots, o_a, E_1$. Let Y be the total number of viewers in movies $o_1, \ldots, o_a$ and let Z be the number of viewers in movies $o'_1, \ldots, o'_b$. By the optimality of `Max-IS`, it follows that the number of viewers in movies $m^1_i, \ldots, m^1_{j-1}$ is no less than neither Y nor Z. Consequently, in the construction of Opt_1, we have deleted movies whose number of viewers is no more than the number of viewers of the movies we have scheduled on channel 1.

Opt_i can be similarly constructed from Opt_{i-1}, which yields the claim. $\square$

We now turn to the evaluation of the performance of the on-line strategies and `Max-IS`, under different distributions of requests and system's load.

4.3 Experimental evaluation

We define the *load* of a `VOD` service as the number of incoming requests in a given time interval. This can be expressed as $x \cdot c$, where x is the *loading factor*. We measure the performance of the different channel allocation algorithms by the revenue they produce. For simplicity, we will assume that the revenue function is simply the number of requests that are honored.

Our simulation experiment consists of 100 channels and 48 intervals (namely, one day, with start times every 1/2 hour). In a "typical" day, we consider 3 values for the loading factor: low, medium and high. The high value corresponds to "prime" time, the medium value corresponds to the intervals around (i.e., before and after) the prime time, and the low value corresponds to the rest of the day. In particular, by normalizing the high value to 1, we choose .3 as the medium value, and .02 as the low value. The first transition from low to medium occurs at interval 22; from medium to high at interval 26; back to medium at interval 29; and back to low at interval 32. This load distribution is depicted in Figure 6.

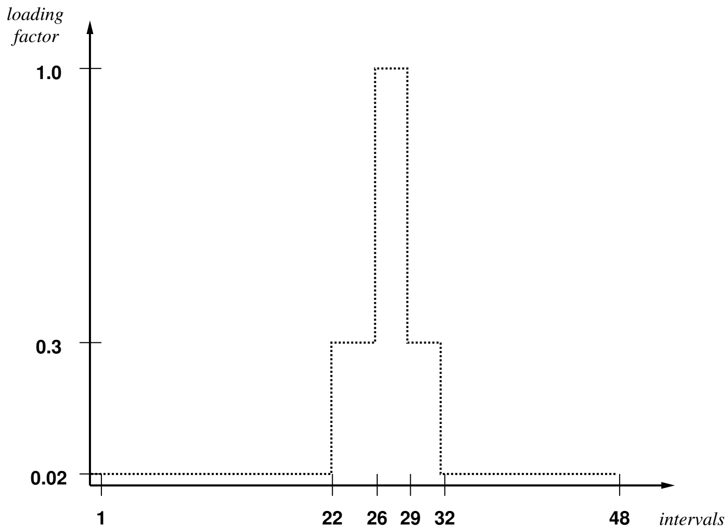

Fig. 6. *Typical load variation*

The request distribution in a given interval was produced as follows. We assumed that each user using the Shared `VOD` service had a fixed probability (0.3

in our experiment) to generate a request for a new movie, without inspecting the list of already requested movies. Otherwise (i.e., with probability 0.7) the user would browse the list. For each movie being listed, we assumed another fixed probability for the movie to be chosen (between 0.001 and 0.02 in the experiment). Should the user decide not to pick a movie from the list, he/she would generate a new request. For example, with a value of 0.1 for the second probability, if the list of movies is of size 5, then with 0.5 probability a new request is generated. This procedure was repeated for each arriving request.

A run of the experiment had two inputs: The number of requests corresponding to a high loading factor, and the probability that a movie from the list is chosen. For each such run, we calculated the total number of viewers for all the on-line channel allocation algorithms: `Greedy`, `Harmonic`, `SF`, and `Max-IS` of Section 4.2

We also evaluated the performance of Full `VOD` and Near `VOD`. In the case of Full `VOD`, we computed the ratio between the number of viewers in `SF` and Full `VOD`. Naturally, this number is greater than 1, and gives an indication of how much more a Full `VOD` user would have to be charged in order for the service provider to accrue the same profit as in `SF`. We call this factor the *cost factor*. In the case of Near `VOD`, we computed the fraction of viewers that must pick a movie from the list in order to achieve the same profit as with the `SF` algorithm. We call this fraction the *hit ratio*.

Part of our experiments are summarized in the figures appearing in the Appendix. Figure 7 shows the performance of all the channel allocation algorithms for Shared `VOD`. In the runs shown in the figure, the number of requests during prime time is 1000, and the probability of choosing a movie from the list is varied from 0.001 to 0.02. Naturally, if users are more inclined to choose movies from the advertised list, then the channels will be shared to a greater extent, and the total number of viewers will be larger. To notice here is that both `Greedy` and `SF` outperform `Harmonic`, and that both are very close to `Max-IS`.

Figure 8 shows the performance of all the algorithms under load variations. In these runs, the probability of choosing advertised movies is fixed at 0.01. There is a clear advantage of `SF` with respect to the other on-line strategies when the load is high. We remark that this is so for the specific load pattern depicted in Figure 6. In other, more uniform, load transition patterns, the performance of `Greedy` is comparable to that of `SF`.

Figure 9 plots the cost factor of Full `VOD`. In the top graph, the probability of choosing an advertised movie is 0.01, and the total number of prime time viewers varies from 1000 to 100000. In the bottom graph, the number of prime time viewers is fixed at 1000, while the probability of choosing an advertised movie is varied from 0.001 to 0.02. The two plots corroborate the expected phenomena that the cost factor will be larger if either the loading factor, or the probability of choosing an already advertised movie, is increased. This is because, in both cases, there will be more sharing of the channels among the users.

Finally, Figure 10 plots Near `VOD`'s hit ratio. The parameters are the same as in the previous case. Here, the hit ratio is larger if the probability of choosing a movie increases because the sharing is larger. On the other hand, the hit ratio decreases when the loading factor increases because there are many rejections by `SF`, and therefore fewer viewers in Near `VOD` need to choose a movie from the list in order to achieve the same revenue.

5 Conclusions and Future Directions

In this paper we formulate the problem of Video on Demand from a resource allocation perspective. We introduce the concept of Adaptive `VOD`, which covers the middle ground between the two popular approaches suggested so far of Full `VOD` and Near `VOD`. We investigate the problem in an on-line fashion, and give a characterization of the competitive ratio of on-line movie scheduling algorithms as a function of the notification time, providing upper and lower bounds.

Video on Demand requires a substantial investment in equipment. Full `VOD` and Near `VOD` take rather extreme views to the functionality/price tradeoff. Inspired by our on-line analysis, we also suggest a concrete scheme, Shared `VOD`, which gives a more balanced tradeoff between functionality and price. In this approach, several users may share the same channels, but the movies are selected by the users. We present a reasonable system design for Shared `VOD` and efficient on-line channel allocation algorithms.

We also compare Shared `VOD` with Full and Near `VOD` under different distributions of requests and load factor, and conclude that under reasonable circumstances, Shared `VOD` would perform better than either approach. Our recommendation is, in reality, to use a service which offers a combination of Shared, Full and Near `VOD`. The design of such combined service, which would address optimization problems such as the number of channels allocated to each of the three services, is a topic for future research.

Another topic of relevance is the pricing structure. The basic purpose of Shared `VOD` is to offer more flexibility than Near `VOD`. However, there is no guarantee in the design presented here that a user's request will be granted, and the user might end up paying for a full service option after his request is rejected.

Thus, a real implementation should provide a compensation for such cases, such as charging less than regular Full VOD users.

Finally, we remark that there are other issues to consider in the design of an actual VOD system, such as the actual behavior of the users, and the selection of Near VOD titles. Some additional natural extensions would be to investigate requests (movies) of different length (cf. [11]), different capacity requirements (e.g., HDTV vs. regular TV quality), dynamically changing capacity requirements (e.g., multi-media documents), and multiple stages of video servers and capacity-limited channels.

Acknowledgments

The authors thank Ernst Biersack and Anna Karlin for helpful comments on preliminary versions of the paper. We also thank the referees for their many technical comments and constructive criticism.

References

[1] S. Albers. The influence of lookahead in competitive paging algorithms. *Proc. 1st Annual European symp. on Algorithms*, LNCS, Springer-Verlag, pp. 1-12, 1993.

[2] B. Awerbuch, Y. Azar, A. Fiat, and T. Leighton. Making Commitments in the Face of Uncertainty: How to Pick a Winner Almost Every Time. *Proc. 28th Annual ACM Symp. on the Theory of Computing*, pp. 519-530, Philadelphia, PA, May 1996.

[3] S. Aggarwal, J.A. Garay and A. Herzberg. Adaptive Video on Demand. *Proc. 3rd Annual European Symp. on Algorithms*, LNCS (979), Springer-Verlag, pp. 538-553, Corfu, Greece, September 1995.

[4] E. Arkin and E. Silverberg, Scheduling jobs with fixed start and end time, *Discrete Applied Math*, 18:1–8, 1987.

[5] C. Bouras, V. Kapoulas, T. Pantziou, and P. Spirakis. Randomized Adaptive Video on Demand. Personal communication. Short abstract in *Proc. 15th Annual ACM Symp. on the Principles of Distributed Computing*, p. 179, Philadelphia, PA, 1996.

[6] R.L. Graham. Bounds on multiprocessing anomalies. *SIAM Journal of Applied Mathematics*, 17:263–269, 1969.

[7] J.A. Garay and I.S. Gopal. Call Preemption in Communication Networks. *Proc. IEEE INFOCOM '92*, 1043–1050, May 1992.

[8] I. Ben-Arroyo Hartman, Optimal k-coloring and k-nesting of intervals, *Proc. 1st Israeli Symposium on Theory of Computing and Systems,* 1992.

[9] C.J. Horton. 110 channels without boundaries: why restrict options? *Communications Engineering and Design*, 19/1:29–30, January 1993.

[10] J. Koegel and A.Syta. Routing of multi-media connections in hybrid networks. *Proc. SPIE - Int. Soc. Opt. Eng.*, 1786:2–10, November 1993.

[11] R.J. Lipton and A. Tomkins. Online Interval Scheduling. *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 302-311, Arlington, VA, January, 1994.

[12] S.M. McCarthy. Integrating Telco Interoffice Fiber Transport with Coaxial Distribution. *Proc. SPIE - Int. Soc. Opt. Eng.*, 1786:23–33, November 1993.

[13] K. Metz. Next generation CATV networks. *Proc. SPIE - Int. Soc. Opt. Eng.*, 1786:184–189, November 1993.

[14] J. McManus and K. Ross. Video on Demand over ATM: Constant Rate Transmission and Transport. *Proc. INFOCOM '96*, Vol. 3, pp. 1357-1362, IEEE Press, 1996.

[15] S. Plotkin. Competitive Routing of Virtual Circuits in ATM Networks. *IEEE J. Selected Areas in Communications*, special issue on Advances in the Fundamentals of Networking (invited paper), pp. 1128-1136, August 1995. `<http://theory.stanford.edu/people/plotkin/routing.html>`.

[16] O. Penttinen, L. Ojala, and T. Pyssysalo. Modelling of an Adaptive Video on Demand Scheduling Algorithm using Stochastic Well-Formed Colored Petri Nets. *Proc. Multimedia and Concurrency Workshop – International Conference on the Applications and Theory of Petri Nets (ICATPN '97)*, pp. 17-30, Toulouse, France, June 1997.

[17] C. Sell. Video on demand internal trial. *Proc. SPIE - Int. Soc. Opt. Eng.*, 1786:168–175, November 1993

[18] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 32:652–686, 1985.

[19] D. Shmoys and J. Wein and D. Williamson. Scheduling Parallel Machines On-line. *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, 131–140, October 1991.

[20] J. Salehi, Z. Zhang, J. Kurose, and D. Towsley. Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing. *Proc. ACM Sigmetrics Conference*, 1996.

[21] E. Torng. A Unified Analysis of Paging and Caching. *Proc. 36th Annual Symp. on Foundations of Computer Science*, pp. 194-203, October 1995.

Appendix: Plots

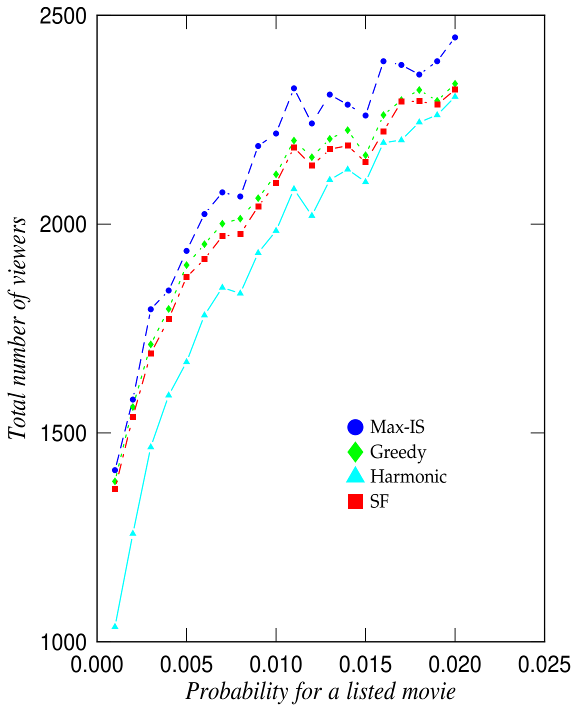

Fig. 7. *Shared VOD for 100 channels*

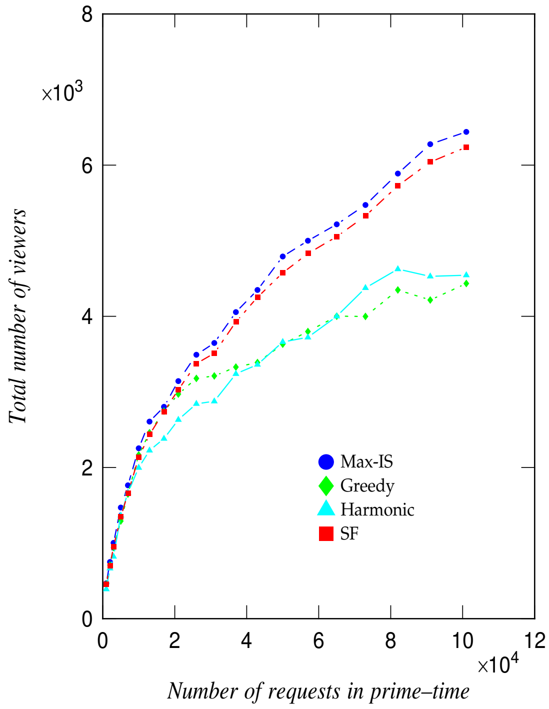

Fig. 8. *Shared VOD for 100 channels*

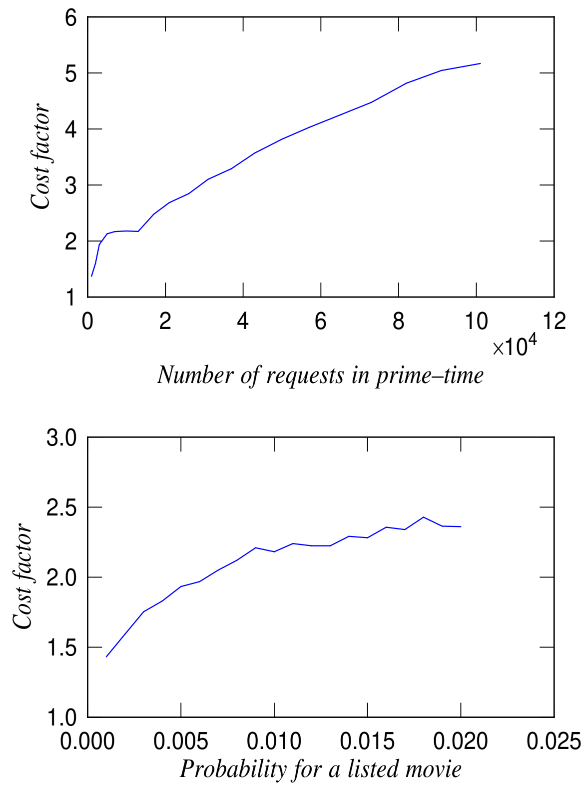

Fig. 9. *Full VOD for 100 channels*

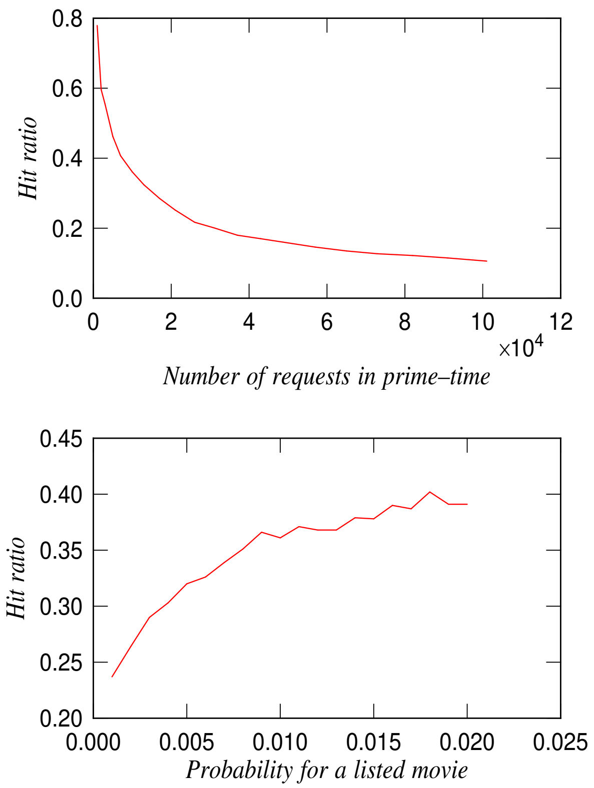

Fig. 10. *Near VOD for 100 channels*