Distributed Prefetching Scheme for Random Seek Support in Peer-to-Peer Streaming Applications

Changxi Zheng
Shanghai Jiaotong University
800 Dongchuan Road
Shanghai, China
cxzheng@ieee.org

Guobin Shen
Microsoft Research Asia
49 Zhichun Road
Beijing, China
jackysh@microsoft.com

Shipeng Li
Microsoft Research Asia
49 Zhichun Road
Beijing, China
spli@microsoft.com

ABSTRACT

Through analysis of large volume of user behavior logs during playing multimedia streaming, we extract a user viewing pattern. The pattern indicates that random seek is a pervasive phenomenon, contrary to the common assumptions that users would watch a video session sequentially and passively in most works on peer-to-peer streaming. We propose to use efficient prefetching to facilitate the random seek functionality. Because of the statistical nature of the user viewing pattern and the ignorance of the users to the content, we argue that the pattern should be used as a guidance to the random seek. Based on the pattern, we set up an analogy between the optimization problem of minimizing the seeking distance and the optimal scalar quantization problem. We then propose an optimal prefetching scheduling algorithm based on the optimal scalar quantization theory. We further propose a hierarchical prefetching scheme to carry out the prefetching more effectively. Real user viewing logs are used to drive the simulations which demonstrate that the proposed prefetching scheduling algorithm and the hierarchical prefetching scheme can improve the seeking performance significantly.

Categories and Subject Descriptors

C.2.4 [**Computer-communication networks**]: Distributed Systems—*Distributed applications*; H.3.5 [**Information storage and retrieval**]: Online Information Services—*Data sharing*

General Terms

Design, Performance

Keywords

Peer-to-peer streaming, prefetching, random seeking

P2PMMS'05, November 11, 2005, Singapore.

1. INTRODUCTION

Multimedia streaming over peer-to-peer (p2p) network has become an active research topic in the past few years, due to the excellent match between the rigorous requirement (e.g., high bandwidth, delay sensitivity etc.) of multimedia content delivery and the resource abundance in a p2p network. Particularly, the local storage of each peer on the network is leveraged as a "cache-and-relay" mechanism [3, 4, 8] for scalable asynchronous multicast. Using this approach, a recipient would "cache" the *most recently* played portion of the feed in a sliding-window fashion. The cached content could then be relayed to the later peers in the system who also request the same feed. Furthermore, the "cache-and-relay" mechanism is extended from caching only the played portion to *prefetching* some future portions using additional bandwidth (besides storage) in paper [12, 13].

In a prefetching protocol, peers prefetch and store various portions of the streaming media ahead of their playing position, which grants peers the ability to overcome the bursty packet loss and the departure of source-peer and to smoothen the playing experience. In a p2p context, as claimed in paper [13], the prefetched portion of content can also serve to others on the network, and ease the content distribution. While it requires the additional bandwidth and storage for prefetching, considering the increasing bandwidth on network and storage capability on local peers nowadays, it actually offers a more desirable tradeoff between quality and cost.

Most of the existing work on p2p-based streaming systems have made an implicit assumption that a user who joined a streaming session would keep on watching till it leaves or fails the session. Hence, the prefetching protocol would prefetch the stream successively, bit by bit without any skips. Unfortunately, based on the analysis of a large amount of real user viewing logs, we found that users usually do *not* play the video successively and passively. Instead, users perform random seek quite frequently. As a result, the real performance of these p2p-streaming systems may suffer from a performance drop in a practical deployment. In paper [9], Jin and Bestavros have shown that asynchronous multicast techniques do not scale as advertised when content is not accessed from beginning to end. In consequence, the random seek functionality should be considered, instead of being overlooked under some unrealistic assumptions, in a practical streaming system because of its significant impact on users' experiences. Specifically, in [17], we proposed a Segment-Tree based control plane protocol for efficient streaming service discovery, with the sup-

port of efficient random seek being an implicit target, in a p2p-based "cache-and-relay" collaborative streaming framework. In this paper, we are further motivated to consider a prefetching scheme, according to the user behavior pattern, in the p2p-streaming framework to better support random seek functionality.

User viewing logs can be utilized in a few ways. Some previous works have already focused on modelling the user behavior pattern. A *Hidden Markov Model*(HMM) was proposed in paper [14] to model the user behavior into several different states during browsing and playing. However, manual training is required for this model to specify which state a user is currently in, which may hinder its deployment when a large amount of movies are to be served. In paper [7], Huang and Chao proposed a scheme to mine the user-behavior pattern from the logs and to prefetch and cache some content on video proxy. In this paper, we also pay our attention to the statistical user behavior pattern, especially the seeking behavior. Different from the previous works, we extract the pattern from user playing logs and use it to guide the prefetching.

Exploiting a prefetching scheduling scheme akin to the optimal scalar quantization, we determine the segments of the video to be prefetched in a strategic way such that the average seeking distance (which will be defined later on) are minimized. For our scheme, we can see the following advantages:

1. Like the previous prefetching schemes, user can employ the prefetching scheme to overcome the bursty network degradation.

2. Due to the statistical nature of large amount of user-behavior logs, the segments prefetched according to the statistical model are more likely to be the desired piece of content of users' seeks. For those already prefetched segments, the additional buffering time is exempted.

3. It provides meaningful guidance for the seek. The reasons are three-fold: first of all, user behavior usually tends to reflect a small-world phenomenon and leads to a Zipf-like distribution [2]; secondly, the seek are generally blind because the users are ignorant to the new video content; and thirdly, the user viewing pattern is actually a rich statistical summarization of other viewers' experiences.

4. It provides an effective fast browsing functionality. This is quite desirable and applicable in the case that there are many movies on a server. User can browse the prefetched slices of the movie to scrabble the skeleton and preview the most wonderful scenes, and determine whether to watch it in detail. The effectiveness results from the fact that the scheduled segments using our algorithm is more representative.

As with many other p2p-streaming works, in this paper, we focused on the prefetching behavior of an autonomous peers, i.e., designing a purely distributed prefetching scheme, and studying its effect on the overall p2p-network where the autonomous peers collaborate. Specifically, even though the ultimate target is to shorten the user seeking delay, instead of studying it explicitly, we study the number of concurrent users which has an implicit impact on shortening the seeking delay, as revealed in other performance modeling work for p2p-network [5, 16].

Table 1: Useful Log Fields

Field	Definition
s-session-id	Internal session ID used by server to track a given client session.
c-playerid	Client GUID. This can be a unique GUID if the user has not disabled unique identification.
c-starttime	Time stamp (in seconds) indicating the start time (position) of this log.
x-duration	Amount of time (in seconds) that the client has been playing the content. Buffered data is excluded.
c-status	Code describing the status of the client.
c-rate	The indicator of user behavior such as play, rewind, fast forward/backward etc.

The paper is organized as follows. In section 2, we provide a detailed description and thorough analysis about the characteristics of user viewing behavior. We then present a model to describe the user behavior pattern and state an algorithm analogized from quantization theory to schedule the prefetching segments in section 3. Next, in section 4, we present a hierarchical prefetching scheme and the cache and replacement policy. Then we evaluate our scheme in section 5 through the simulations driven by real user viewing logs. Finally, we conclude the paper and discuss some future works in section 6.

2. USER VIEWING PATTERN

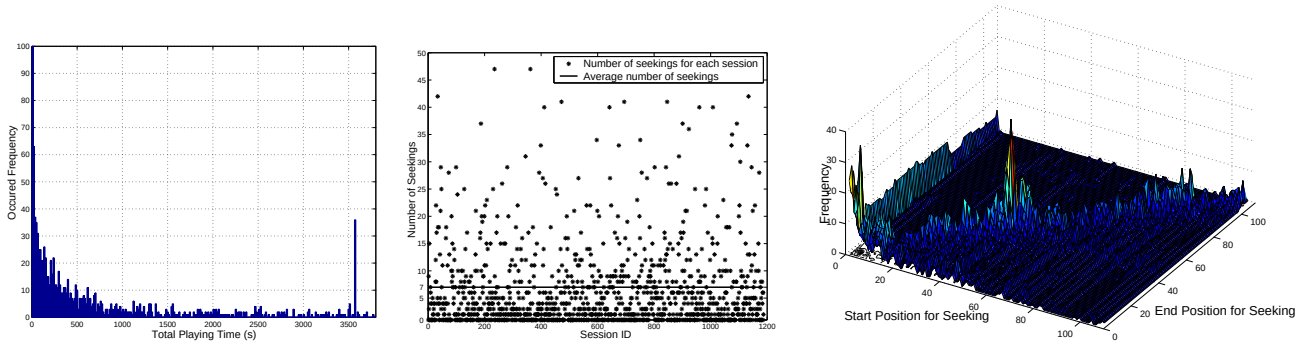
In this section, we perform in-depth analysis of user viewing behavior through a large amount of user viewing logs.

2.1 Log collection

We collected the user viewing logs on the multimedia streaming server at *Shanghai Jiaotong University, China* from Nov. 16, 2004 to Dec. 19, 2004. There are hundreds of movies on that server to for the entertainment purpose for the students living on campus. Thousands of viewing sessions occurred daily from that server. Throughout this paper, by a session, we mean a user watching a movie. The server employs *Microsoft Windows Media Server*, and produces detailed logs for each session. We collected more than 600 thousand log items about all of those movies and about 150 thousand individual sessions throughout the more than one month duration. This volume of logs, we believe, is adequate for mining out a convincible pattern of user viewing behavior.

On the Microsoft Windows Media Server, a log item is generated whenever one of the client events occurs, such as session setup, session teardown, seek, fast forward or backward, etc. Each item consists of several fields. Those useful for the user viewing behavior analysis are listed in Table 1.

In this paper, we mainly focus on the seeking rather than the other behaviors such as rewinding, fast forward/backward etc. A session can be uniquely identified using the fields *s-session-id* and *c-playerid*. We exclude the error status and the case of rewinding/fast forwarding/fast backwarding ac-



(a) Accumulated viewing duration (b) Number of random seeks (c) Seeking frequency vs positions

Figure 1: Plots of statistical features of user viewing behavior for a typical movie: (a) The accumulated view duration; (b) The number of random seeks; (c) The relation of seeking frequency and occurring positions.

cording to the fields *c-status* and *c-rate*, respectively. Finally, the fields *c-starttime* and *x-duration* reflect the playing and seeking behaviors.

2.2 Characteristics of user viewing behavior

After distilling the large amount of user viewing logs, we extract some useful information such as the accumulated viewing duration per movie, the number of random seeks cross sessions, and the seeking frequency vs positions per movie etc. These features of a typical session are plotted in Figure 1.

From the plots in Figure 1, we can reach the following observations/conclusions:

1. Users rarely view the movie from the beginning to the end. Instead, for a single movie, the total playing time of a user is quite limited and tends to be short. Figure 1(a) shows the histogram of the accumulated user playing time for a single movie. It is apparent that many sessions only last a very short period of time. The result is led to mainly by the following two causes: 1) the user browse a movie and tear down the session when find the movie does not fit his/her taste; 2) the user only wants to watch a special portion of the movie. Of course, from figure 1(a), we can also see a peak near the end of the movie, which implies there are still many users watched almost the whole movie. Nonetheless, there are still seeks since the peak does not appear at the end point, i.e., the accumulated viewing duration is still less than the movie length.

2. Most users always perform some random seeking, more or less, during the session, as can be clearly seen in Figure 1(b). This is counter to the assumption in most of existing p2p-streaming work. The average amount of seeking is about 7 times. This observation actually fits our own viewing experience well: one would always like to skip the boring scenes, to preview the climax or to review some meaningful dialogues. All of these render the seeking operation to be a pervasive phenomenon.

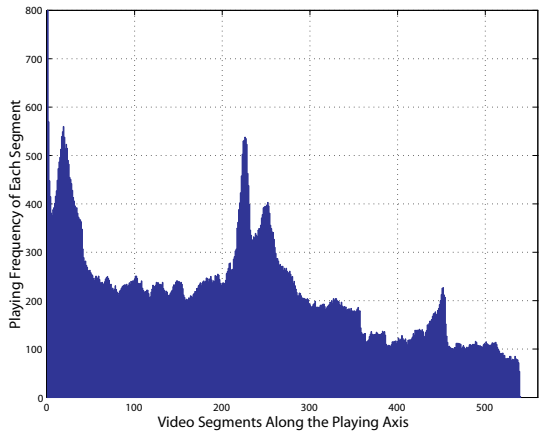
3. Another remarkable point is that the seeking is always aimless. We divide a movie uniformly into segments, each of which lasts about 50 seconds, and accumulate the occurring frequency for seeking from a source segment y to a destination segment x. There are two obvious trends: 1) Many of the seeks are forward-looking yet aimless, or in other words, browsing-like behavior, as can be seen from the x axis in Figure 1(c). This is due to the ignorance of the user to the movie; 2) for many of the seeks, the source position and the destination position are quite close, as can be inferred from the rigid along the line $y = x$ in Figure 1(c). This is generally due to the fine tune of a previous seek to reach the exact desired viewing point or fast forward/backward behavior. These observations are also coincident with our common experiences.

2.3 Playing and seeking pattern

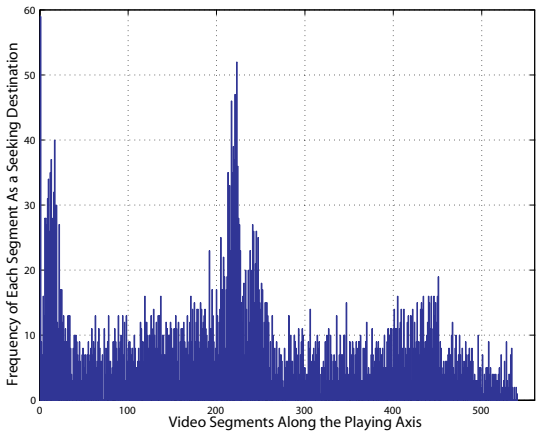
Having identified that seeking is such a pervasive phenomenon, now we want to extract some useful patterns such that we can utilize them for better support of the random seek functionality.

We choose some the popular movies from the server, uniformly divide each of the movies into segments, each of which lasts about 10 seconds. We will use such segment as the prefetching unit later. Then we accumulate the times for each segment being played. Figure 2(a) shows the playing frequency for one of the movies, which is about 5400 seconds in length. According to the figure, the first peak appears at the offset of about 200 seconds, which is the beginning of the movie right after the prelude. The second peak comes in the period from 2100 seconds to 2500 seconds. This portion is the most attractive scenes in this movie. The playing pattern is quite intuitive since people always like to watch the most wonderful portion of a movie. Wonderful scenes are always appreciated by most audience and leads to a consensus on the popularity of a movie. Note that the two peaks are consistent with those appeared in Figure 1(c).

Now let's pay our attention to the seeking pattern. As with the above analysis, we divide the movie into segments. Since we are more care about the destination of a seek operation, we calculate the accumulated frequency of seeking destinations. In other words, we are more interested in the marginal distribution of seeking destination rather than the joint distribution of both the seek source position and the seeking destination even though it also provides useful insights. To our surprise, as shown in 2(b), the accumulated



(a) Playing frequency



(b) Seeking frequency

Figure 2: Playing/seeking pattern: (a) Playing pattern for a movie; (b) Seeking pattern for a movie;

Table 2: Notations

Notation	Definition
$\mathcal{P}_p(x)$	Playing frequency of each segment x
$\mathcal{P}_s(x)$	The frequency of seeking to the segment x
L	Number of segments that should be prefetched
$\boldsymbol{g}$	A specific prefetching scheme.
$d(x, \boldsymbol{g})$	The seeking distance for a destination segment x under a prefetching scheme $\boldsymbol{g}$.
$D_s(\boldsymbol{g})$	Expected seeking delay under a prefetching scheme $\boldsymbol{g}$.
$p(x)$	The probability of seeking to segments x.
$\mathcal{B}_l$	The l^{th} partition region in the video playing time span.
g_l	The l^{th} prefetched segment.

seeking frequency distribution seems highly correlated to that of playing frequency. That reflects that people want to seek directly to the hot spots of a movie. Note that this pattern actually presents an opportunity that we may shorten the seeking delay (i.e., increase the hit ratio) according to the seeking pattern.

3. PREFETCHING SCHEDULING

In this section, we start with the basic pattern model. We then formulate the optimal prefetching scheduling problem and apply the quantization theory to solve the problem. The notations used in this paper are listed in Table (2).

3.1 Pattern Model

As mentioned above, we divide the video into segments as the prefetching units in our scheme. Intuitively, the playing pattern can be easy modelled as $\mathcal{P}_p(x)$ where x represents the specified segment in the movie. But for the seeking pattern, we have two choices, namely $\mathcal{P}_s(y, x)$ and $\mathcal{P}_s(x)$. The former represents the frequency of a seek operation from a source segment y to a destination segment x. It can also be interpreted as the frequency of a seek operation to a destination segment x where the segment y has been played previously. All in all, this model considers the seeking dependency between two segments. However, it is not practically tractable since it requires a large amount of logs to extract a meaningful 2-D pattern function. If there are 500 segments in a movie, it need the logs to cover the whole 250,000 samples. In fact, we cannot extract a convincing pattern from our one-month collections. On the other hand, the latter model is actually the marginal distribution of $\mathcal{P}_s(y, x)$. It is quite simple and easy to obtain. It also provides a good capture of general user viewing patterns and reflects good insights. As will be demonstrated later on, a remarkable result can be obtained based on this model, which proves the validity of the model.

In short, we use the two pattern functions $\mathcal{P}_p(x)$ and $\mathcal{P}_s(x)$ to represent the playing and seeking frequency, respectively.

3.2 Optimization problem statement

Knowing the playing and seeking patterns, we state the problem we are going to solve formally in this section. In general, we want to find a scheduling algorithm to optimally determine which segments should be prefetched. Before preceding further, we need a quantifiable metric so as to tell different scheduling algorithms. We use the *seeking distance* as the metric which is defined as the minimum distance between the desired seeking destination and resulting position of a scheduled segments. The rationale of this metric is as follows: as argued before, the user viewing pattern should be used as a guidance of a random seek operation. On the other hand, the users' seeking requests are usually *noisy* because they are ignorant to the movie content. Suppose we can find some most representative segments from the movie, then when different users want to seek to different destinations, we can safely schedule them to prefetch the nearest representative segments. This "trick" (or guidance) can be viewed as a filtering process to remove the noise in users' seeking requests. Since the user viewing pattern is a statistical summarization of many users, the most representative segments should have minimum average distance to all users' seeking requests.

Given a prefetching scheme $\boldsymbol{g}$ and a seeking destination

x, then the expectation of the seeking distance is:

$$D_{\boldsymbol{g}} = E\{d(x, \boldsymbol{g}(x))|x \in \mathcal{L}\} = \int_{x \in \mathcal{L}} d(x, \boldsymbol{g}(x))p(x)dx \quad (1)$$

where $d(x, \boldsymbol{g}(x))$ is the seek distance which obviously depends on the prefetching scheme $\boldsymbol{g}(x)$ and the seeking destination x and is a function of them.

The probability $p(x)$ is quite difficult to determine since it relies on several varying factors, including the individual tastes, the segments already played by the user, the playing state(browsing/seeking/watching), etc. How to model $p(x)$ is the focus of user behavior modeling works [1,14]. Here we want to establish $p(x)$ from the user viewing behavior logs. In other words, $p(x)$ is represented as a function of $\mathcal{P}_p(x)$ and $\mathcal{P}_s(x)$.

$$p(x) = f(\mathcal{P}_p(x), \mathcal{P}_s(x))$$

We argue that the synthesis function f depends on different application scenarios. We tried several synthesis schemes, and finally, we simply employ

$$p(x) = f(\mathcal{P}_p(x), \mathcal{P}_s(x)) = C \times \mathcal{P}_p(x) \times \mathcal{P}_s(x) \quad (2)$$

as the synthesis function that works well, where C is a normalization factor. As an extension work, we can design different kinds of synthesis function, and provide an interface to user, so that them can customize it as the individual desire.

Now the target of the optimization problem is to find an optimal prefetching scheme $\tilde{\boldsymbol{g}}$ to minimize the expected delay.

$$\tilde{\boldsymbol{g}} = \underset{\boldsymbol{g}}{argmin}\{E\{d(x, \boldsymbol{g}(x))|x \in \mathcal{L}\}\} \quad (3)$$

3.3 The proposed solution

In this subsection, we first present two intuitive solutions and then briefly review the quantization theory based on which our optimal solution is derived

3.3.1 Intuitive solutions

The most intuitive and simplest way for which segments to be prefetched is to uniformly distribute them throughout the movie. But this kind of decision is usually far from optimal since it ignore the user viewing pattern completely. Another intuitive solution that considers the user viewing pattern is a greedy "inverse water-filling" algorithm. As illustrated in Figure 3, suppose there is a horizontal line moving downward. It will intersect with the curve of probability function $p(x)$. Since the high probability areas are intersected earlier than low probability areas, one can treat the intersection to be the portion need to be prefetched. The line will keep on moving down until the intersection is large enough (i.e. L prefetched segments are found).

However, this seemingly optimal algorithm is not a globally optimal one. It results in a local optimum because only the most popular segments are to be prefetched. Some other sub-popular segments can never be prefetched. As an example, in Figure 3, there is a peak at offset t, which represents a sub-popular (not most popular but still popular) segment. If the inverse water-filling algorithm are to be used, it can never be prefetched since the prefetching always concentrates on the most popular segments. However, it is not necessarily good to prefetch all the most popular segments. As argued in paper [11], in order to shorten the delay, only

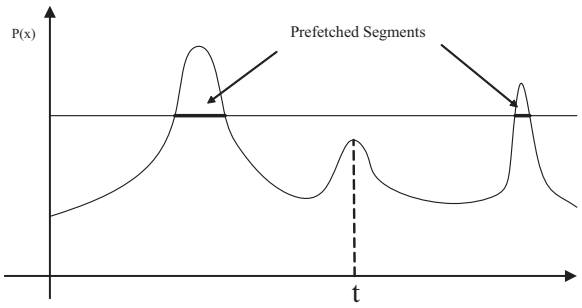


Figure 3: Greedy solution for the optimization prefetching problem and its shortage.

a portion of the video need to be download/prefetched.[1] Moreover, we are more interested in studying a scheme so that it offers best performance for the whole network, instead to benefit some peers while sacrifice the others. With these considerations, we found the problem are indeed very similar to the optimal quantization problem.

3.3.2 Quantization theory based solution

Before presenting our solution, we briefly review the *Scalar Quantization Theory* [15] widely used in image/video compression.

In scalar quantization, each sample in a source signal is quantized into one of the reconstruction values in a predesigned codebook. A quantizer is characterized by the quantization level, $\bar{L}$, the boundary values, $b_{\bar{l}}, \bar{l} = 0, 1, \ldots, \bar{L}$ and reconstruction values, $g_{\bar{l}}, \bar{l} = 1, 2, \ldots, \bar{L}$. The boundary values establish the partition regions $\mathcal{B}_{\bar{l}} = [b_{\bar{l}-1}, b_{\bar{l}})$. Letting $\bar{\mathcal{L}} = \{1, 2, \ldots, \bar{L}\}$, the quantization function is described by:

$$Q(f) = g_{\bar{l}}, \quad \forall f \in \mathcal{B}_{\bar{l}},\ \bar{l} \in \bar{\mathcal{L}}$$

The distortion of a quantizer, $\boldsymbol{q}$, is defined accordingly as the average distance between the original and the quantized samples.

$$D_{\boldsymbol{q}} = E\{d(f, Q(f)\} = \int_{f \in \mathcal{B}} d(f, Q(f))p(f)df \quad (4)$$

The optimal scalar quantization problem is to find the partition regions $\mathcal{B}_l$ and the reconstruction values g_l for a specified quantization level L such that the distortion metric defined in formula (4) is minimized. That is, to find a quantizer $\boldsymbol{q}$ such that

$$\tilde{\boldsymbol{q}} = \underset{Q}{argmin}\{E\{d(f, Q(f))|f \in \mathcal{B}\}\} \quad (5)$$

From the optimal scalar quantization theory, it is stated that the reconstruction values lie at the centroids of the partition regions between the boundary values. The boundary values and the reconstruction values should satisfy the *nearest-neighbor condition* and *centroid condition* described as follows:

$$\mathcal{B}_{\bar{l}} = \{f : d(f, g_{\bar{l}}) \leq d(f, g_{\bar{l}'}), \forall \bar{l}' \neq \bar{l}\} \quad (6)$$

$$g_{\bar{l}} = E\{f|f \in \mathcal{B}_{\bar{l}}\} \quad (7)$$

[1]In our opinion, it is good enough for the length of the prefetched portion to be larger than (so as to hide) that of the service discovery delay.

Table 3: Lloyd algorithm for prefetching scheduling in a given region $\mathcal{B}$

prefetching_sche(L, $\mathcal{B}$, $p(x)$)
  divide region $\mathcal{B}$ uniformly into L sub-regions.
  For each sub-region: $g_l \leftarrow \sum_{x\in\mathcal{B}_l} xp(x)$
  Seeking Delay: $D \leftarrow \frac{1}{K}\sum_{l\in\mathcal{L}}\sum_{x\in\mathcal{B}_l} d(x, g_l)$

  do
    $D_{last} \leftarrow D$
    foreach sub-region
      $g_l \leftarrow \sum_{x\in\mathcal{B}_l} xp(x)$

    foreach sub-region
      update the boundary value: $b_l \leftarrow (g_l + g_{l+1})/2$

    Seeking Delay: $D \leftarrow \frac{1}{K}\sum_{l\in\mathcal{L}}\sum_{x\in\mathcal{B}_l} d(x, g_l)$
  while $abs(D - D_{last}) \geq V_{threshold}$

  return $g_l \quad l = 1, 2, \ldots L$

Table 4: Hierarchical prefetching scheme

seek_to(x)
  while region_stack.top() does not cover x
    foreach segment **in** region_stack.top()
      segment.set_to_be_replaced_out()
    region_stack.pop()
    level_stack.pop()
  end while

  if region_stack.size() < max_hierarchy_level
    $\mathcal{B}$ ←region_stack.top().subregion(x)
    $L \leftarrow \lambda$(level_stack.top())
    prefetching_sche(L, $\mathcal{B}$, p)
    region_stack.push($\mathcal{B}$)
    level_stack.push(L)
    Order_Prefetching_Segments()
    Start_Prefetching()
  end if

Return to our problem. Our target is to minimize the seeking delay defined in equation (1). Yet the target of optimal scalar quantizer design is to minimize the distortion defined in equation (4). Comparing the two equations, we can easily see the analogy between them. This analogy is further confirmed if we compare the equations (3) and (5). If the seeking delay function $d(x, G(x))$ can be mapped to the distortion measure $d(f, Q(f))$, then a sample f will be mapped to a seek operation and the reconstruction values, g_l, can be mapped to the segments to be prefetched. Furthermore, we can also bring in the concept of *boundary values* and *partition regions* into our problem. The partition regions come from the division of the whole playing period, each of which contains a prefetched segment. When users feel like to seek to a partition region, the prefetched segment in that region is a *suggestion* for the seeking. And when users browse the video, the prefetched segment can be used to represent the region that it belongs to.

The optimal scalar quantization problem can be elegantly solved using the classical *Lloyd Algorithm* [6,10]. According to above reasonings, we can readily apply the Lloyd algorithm to solve our optimization problem, i.e., to optimally determine which segments should be prefetched for a given prefetching amount, L. The final prefetching algorithm is given in Table 3. The synthetic probability function $p(x)$ is used as the probability distribution function in Lloyd algorithm.

4. HIERARCHICAL PREFETCHING

Having determined the optimal representative segments above, in this section, we focus on how to prefetch the determined segments. From the perspective of individual peer, it needs a scheme about the prefetching order and also a cache replacement policy due to possible limitation on storage.

Until now, the only parameter undetermined in our solution described in section 3.3 is the prefetching level, L, which means how many segments should be prefetched. On the one side, we expect to prefetch the segments as many as possible, i.e., the larger L, the better. But on the other side, it is limited by the available bandwidth and local storage. Clearly, L controls the prefetching precision and the cost. Therefore, we choose to let L be a configurable system parameter, even though we do provide some intuitions for determination of L. To provide a better tradeoff, we propose a hierarchical prefetching scheme with a simple cache replacement strategy.

Initially, the initial value of $L^{(0)}$, ($L^{(0)} = L$) is chosen according to the video length, number of local peaks in $p(x)$ and available local storage. Then the $L^{(0)}$ is used to determine which segments should be prefetched. The corresponding *support region* $R^{(0)}$ (a region on which to apply the optimal prefetching scheduling algorithm), is the entire video. The $L^{(0)}$ determined segments are ordered according to their popularity in $p(x)$: the segments with higher popularity are prefetched with higher priorities. Note that we also consider the phenomenon that users always like to seek forward rather than backward, and hence the ordering is biased for forward seeking.

Now suppose a seek operation is occurred, it jumps to a new position, which supposedly belongs to the partition region, $\mathcal{B}_l^{(k)}$, at hierarchy level k and region l. Now the segments that are scheduled and unfinished before the new seek are reordered according to the new position. Meanwhile, the current partition region $\mathcal{B}_l^{(k)}$ is set as the new support region, $R^{(k)}$. The optimal prefetching scheduling algorithm in section 3.3 is applied recursively to determine $L^{(k)}$ segments from $R^{(k)}$. In our hierarchical prefetching scheme, we use a function to determine the number of segments to be prefetched from one hierarchy level to the next level, namely $L^{(k)} = \lambda(L^{(k-1)})$. Then the $L^{(k)}$ new segments are ordered and prefetched. Note that we do not impose any limitation here for the function $\lambda(\cdot)$. However, to save bandwidth and storage in practice, the function decreases as hierarchical level increases. In our experiments, we simply let $L^{(k)} = \lambda(L^{(k-1)}) = L^{(k-1)}/2$ for storage considerations as will be elaborated later on.

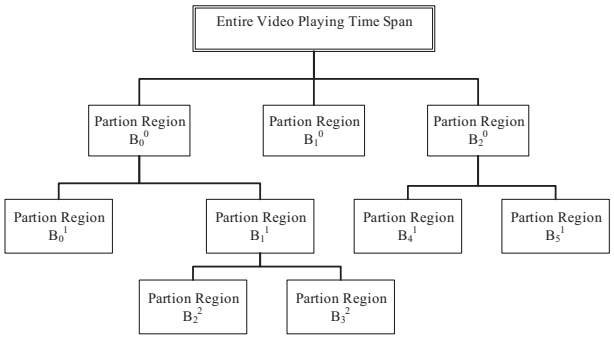


Figure 4: An example for hierarchical prefetching.

The pseudo-code of the hierarchial prefetching scheme is listed in Table 4. The hierarchy level is controlled by a system parameter, K.

For more clarification, we illustrate the hierarchical prefetching scheme in Figure 4. Initially, the prefetching level is three, so it determines three partition regions, namely $\mathcal{B}_0^{(0)}$, $\mathcal{B}_1^{(0)}$ and $\mathcal{B}_2^{(0)}$, each of which contains a prefetched segment. When a seek operation occurred with the destination being in region $\mathcal{B}_0^{(0)}$, which now is the support region. It partitions the region $\mathcal{B}_0^{(0)}$ into two sub-regions $\mathcal{B}_0^{(1)}$ and $\mathcal{B}_1^{(1)}$, and schedules two segments to be prefetched in the two sub-regions, respectively. Then another seek operation occurred, this time the destination is in the sub-region $\mathcal{B}_1^{(1)}$. Same operation will be applied onto sub-region $\mathcal{B}_1^{(1)}$ again. That is, it partitions the $\mathcal{B}_1^{(1)}$ into $\mathcal{B}_2^{(2)}$ and $\mathcal{B}_3^{(2)}$ and schedules another two segments to be prefetched for them. Now a new seek operation that seeks to a destination in $\mathcal{B}_2^{(0)}$ happened. Then it will partition the region $\mathcal{B}_2^{(0)}$ into two sub-regions $\mathcal{B}_4^{(1)}$ and $\mathcal{B}_5^{(1)}$ and apply scheduling accordingly. Suppose now the storage is not enough, we will replace the segments contained in the $\mathcal{B}_0^{(1)}$, $\mathcal{B}_1^{(1)}$, $\mathcal{B}_2^{(2)}$ and $\mathcal{B}_3^{(2)}$, which are the child regions of $R^{(1)}$, with new coming segments.

As stated before, in the hierarchical prefetching scheme, $L^{(k)} = \lambda(L^{(k-1)})$ is used to determine the number of segments from one hierarchy level to the next, mainly for storage reasons. Given the maximum hierarchy depth, K, the total number of segments that are to be prefetched is at most, we have

$$\begin{aligned} S =& L^{(0)} + L^{(1)} + \cdots + L^{(K)} \\ =& L^{(0)} + \lambda(L^{(0)}) + \cdots + \lambda^{(K-1)}(L^{(0)}) \end{aligned} \quad (8)$$

where $L^{(0)} = L$ is the system parameter that controls the number of segments to be prefetched initially. Obviously, the required local storage is completely controlled by the parameters K and $L^{(0)}$. Specifically, if $L^{(k)} = L^{(k-1)}/2$, we have

$$S < \lim_{K\to\infty} \frac{2^K - 1}{2^{K-1}} L^{(0)} = 2L$$

Finally, we want to point out that the hierarchical prefetching scheme is independent of the proposed prefetching scheduling algorithm. It can be combined with any other scheduling algorithm.

5. EXPERIMENTAL RESULTS

In this section, we show some simulation results to demonstrate the advantages of our prefetching scheduling algorithm and the hierarchical prefetching scheme for both the seeking accuracy and the network utilization.

5.1 Evaluation Methodology

Unlike the simulations in others' work, we use the real user viewing behavior logs to drive the simulations, which we believe will reveal more realistic insights. Specifically, we divide the user viewing logs into two separate set, namely training set and testing set. The logs in the training set is used to mine the user viewing patterns as described in Section 3. The logs in the test set are used to generate new user seeking requests.

We choose some popular movies from the video server as our research objects. In the one-month collection, we get more than 8000 logs for each of these movies. We randomly assign around 60% of them into the training set and the rest is grouped into testing set.

5.2 Prefetched segments distribution

We choose the movie that lasts about one and a half hours. Each segment lasts 10 seconds. The initial number of partition regions is 10. We use the Lloyd algorithm to determine the prefetched segments on the top-level prefetching. The results are shown in Figure 5. We overlay the distribution of the prefetched segments onto the synthetic seeking probability function $p(x)$ according to equation (2). The serif thin dotted lines represent the boundary of each partition region, and the serif dashed lines represent the segments to be prefetched.

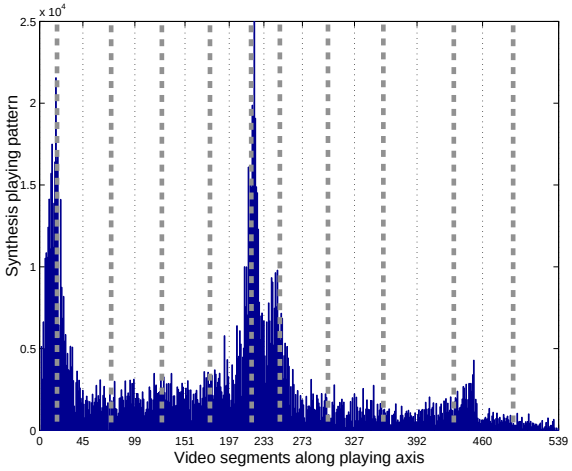
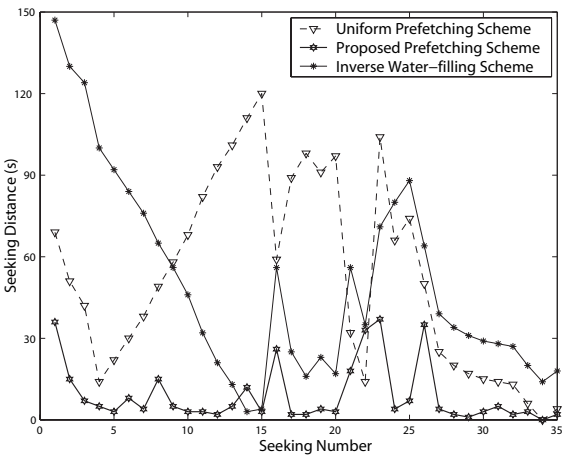
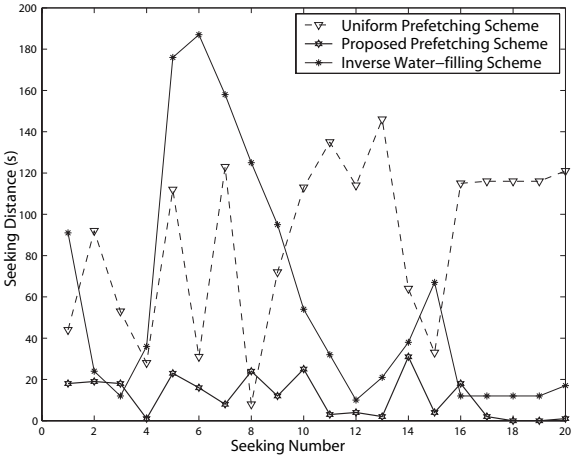


Figure 5: Distribution of prefetched segments using proposed quantization theory based prefetching scheduling algorithm.

As expected, we can see that: (1) Not only the segments with highest seeking probability are prefetched, but the sub-popular segments are also prefetched. This prevents the shortage of the greedy inverse water filling algorithm in section 3.3.1. (2) The distances between the prefetched segments are not uniform and reveals the underlying user viewing pattern. This is in sharp contrast to the intuitive uni-



(a) Seeking Distance of User 1



(b) Seeking Distance of User 2

Figure 6: Comparison of seeking distance of two typical users using different prefetching schemes.

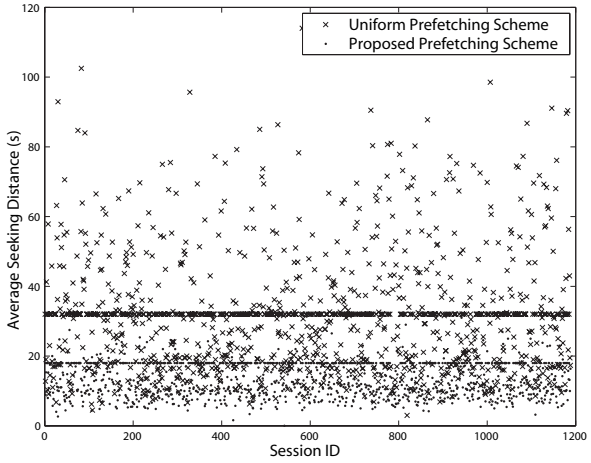


Figure 7: Average seeking distance cross sessions

form prefetching scheme which is not shown in the figure. We are prefetching more segments around the more popular portion in the movie.

5.3 Seeking Distortion

In this section, we show the experimental results of users' seeking. The movie used in this experiment lasts about one and a half hours. Each segment lasts 10 seconds. The initial number of partition region is eight, and hence it has at most three hierarchy levels (due to our bisection strategy in the hierarchical prefetching scheme).

We use the seeking distance that is defined previously in section 3.2 as the metric to evaluate the effectiveness of our prefetching scheme. If any of the prefetched segments cover the seeking destination, then the seeking distance is zero. If we take into account the guidance on users' behavior of the prefetched segments, the effort would be much more remarkable. In order to demonstrate the advantage of our scheme, we compare it with the two intuitive schemes: One is the inverse water-filling scheme and the other one is the uniform scheme that selects the prefetching segments uniformly during the region, as stated in section 3.3.1.

Firstly, we randomly select some sessions from the logs and track their seeking operations to measure the delay. The results of the seeking distance are shown in Figure 6 for two users. Clearly, the seeking distance of our scheme is much smaller than the uniform prefetching scheme. That implies that it indeed provides meaningful guidance and helps to prevent blind seeking. Although the inverse water-filling scheme can also significantly shorten the seeking distortion for some seeking destinations, it results in large seeking distortion at some other positions. This is because the inverse water-filling scheme only benefits the seeking to most popular positions, while complete neglect the seeking request to a less popular destinations. As a result, the seeking distance fluctuates violently.

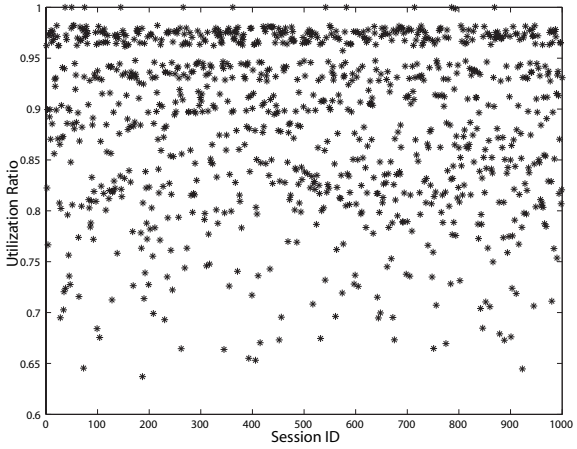
Secondly, we evaluate the average seeking distance across different viewing sessions of the same movie, which reflects the overall user experiences for random seeking. As shown in Figure 7, the average seeking distance in our scheme is clearly smaller than the uniform prefetching scheme.

With above two experiments, we conclude that our proposed quantization theory based prefetching algorithm can shorten the seeking delay remarkably for all the individuals and therefore is quite beneficial to the overall user experience.

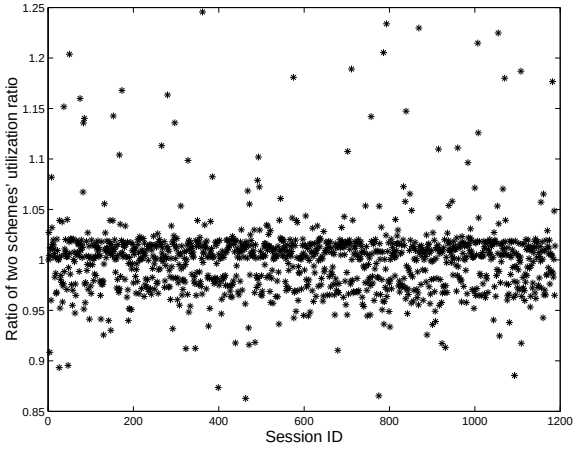
5.4 Utilization Ratio

In this section, we show that our scheme shorten the seeking distortion without much extra expense of bandwidth or local storage. It is quite possible that some segments are prefetched but are not played since the user may seek away prior to the prefetched point. In this case, these segments would not be utilized and is considered a waste of resources. In order to evaluate that, we define a metric *utilization ratio*, which is the ratio of the amount of played data to the amount of download data. As shown in Figure 8(a), the ratio is typically around 0.96 with an average value of 0.918.

We further compare the utilization ration of our scheme with the traditional prefetching scheme that prefetches the



(a) Utilization Ratio



(b) Comparison of two schemes' utilization ratio

Figure 8: Utilization ratio and comparison: (a) The utilization ratio for the proposed hierarchical prefetching scheme; (b) The ratio of the utilization ratio for the successive prefetching scheme and the proposed one.

future segments successively. To be more intuitive, we plot the ratio between the two utilization ratios in Figure 8(b). Most of the samples are concentrated near or under the line $y = 1$. This indicates that the utilization ratio of our scheme is slightly better. This in partial confirms that the proposed prefetching scheduling algorithm indeed provides more meaningful seeking guidance since users are more satisfied with the prefetched content. Moreover, we are measuring the utilization ratio for the whole viewing session where typically around ten seek operations would happen. The advantage of our scheme will be significantly exaggerated if we only measure around the seeking points. However, considering the traditional successive prefetching scheme does not aim at supporting random seek, we feel it is unfair to make such a comparison.

5.5 Impact on the overall network

As stated before, even though the ultimate target is to shorten the user seeking delay, instead of studying it explicitly, we study the number of concurrent users which has an implicit impact on shortening the seeking delay. Specifically, we show the distribution of the peers that have prefetched or scheduled to prefetch for all the segments of a typical movie.

Assume peers arriving to the network following a *Poisson Distribution* with a density of 5 peers per second. Once a peer arrives to the network, it starts to playing according to the real user-behavior logs. The movie length, the segment length, the initial number of partition regions and the hierarchy level are all consistent with the above experiments.

We record the number of peers in the each of the partition regions at a moment. According to our prefetching algorithm, all the peers in the same partition region would try to prefetch the same segment. Therefore, on a p2p-network, the more peers in a partition region, the easier it is to find the more peers to prefetch from or to collaborate with. We plot the peer number distribution in Figure 9 for the top-level and second-level partition regions in our hierarchical prefetching scheme and for both the popular and the sub-popular segments as well. We can observe that the higher the partition region is, the more peers this region is

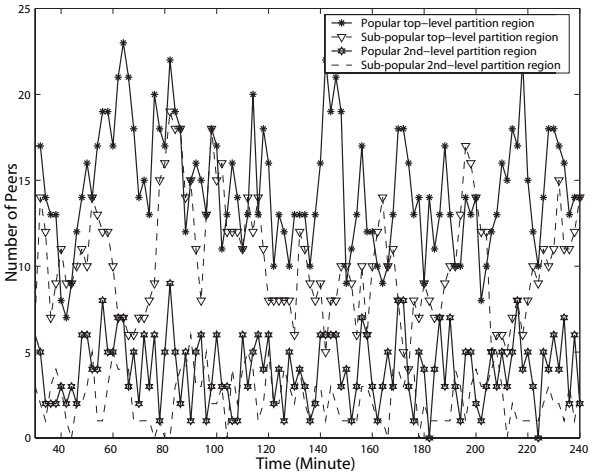


Figure 9: Distribution the number of prefetching peers on a Peer-to-Peer Network

containing. And naturally, there are more peers in a popular region than that in a sub-popular region. An interesting conclusion that can be drawn from the figure is that our hierarchical prefetching scheme actually favors coarse seeking, which achieve one of our initial design goal to provide browse-like functionality.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we performed thorough examination of the large amount of user viewing behavior logs and extract a user viewing pattern. Our analysis reveals that random seek is a pervasive phenomenon. We then propose to use efficient prefetching to better support random seek functionality. We argue that the pattern should be used as a guidance to random seek since the user is in general ignorant to the movie content and the pattern is a statistical summariza-

tion of other users view experiences. We propose an optimal prefetching scheduling algorithm, which is based on the optimal scalar quantization theory, and a hierarchical prefetching scheme for efficient and effective prefetching. Extensive simulation experiments were conducted where real user viewing logs are used to drive the simulations. Experimental results demonstrates that the proposed prefetching scheduling algorithm and the hierarchical prefetching scheme can significantly improve the seeking performance.

In the paper, we are using the user viewing behavior logs extensively. It is generally easy to collect such logs when a central streaming server exist. How to collect the logs in a pure peer-to-peer system is a challenging topic. Moreover, how to efficient publish/discover the prefetched content is also an interesting topic. Finally, the prefetching scheduling algorithm developed in this paper only considered user viewing logs and is content agnostic. How to combine the user viewing pattern and the results of content analysis is also a topic worth further pursuing. All these three topics are our future works.

7. ACKNOWLEDGMENTS

The authors would like to thank Kaida Jiang, the administrator of network and multimedia center at Shanghai Jiaotong University for his kind help on collecting the logs.

8. REFERENCES

[1] T. Akiba and H. Tanaka. A bayesian approach for user modelling in dialogue systems. Technical Report Dept. of Computer Science, ISSN 0918-2802, Feb. 1992.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distribution: Evidence and implications. In *IEEE Proceedings of the INFOCOM*, 1999.

[3] Y. Cui, B. Li, and K. Nahrstedt. ostream: Asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications*, 22(1), Jan. 2004.

[4] D. Eager, M. Vermon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. Knowl. Data Eng.*, 13:742–757, 2001.

[5] Z. Ge, D. R. Figueiredo, S. Jaiswal, J. Kurose, and D. Towsley. *Modeling peer-peer le sharing systems.* In *IEEE Proceedings of the INFOCOM*, 2003.

[6] A. Gersho and R. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Press, Boston, 1992.

[7] C.-M. Huang and T.-H. Hsu. A user-aware prefetching mechanism for video streaming. *World Wide Web*, 6(4):353–374, 2003.

[8] S. Jin and A.Bestavros. Scalability of multicast delivery for non-sequential streaming access. In *IEEE ICDCS'03 Workshop on Data Distribution in Real-Time Systems*, 2003.

[9] S. Jin and A. Bestavros. Scalability of multicast delivery for non-sequenticl streaming access. In *Proc. of SEGMETRICS'2002: The ACM International Conference on Measurement and Modeling of Computer System*, 2002.

[10] P. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inform. Theory*, IT-28:127-35, Mar. 1982.

[11] S. Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *IEEE Proceedings of the INFOCOM*, Apr. 1999.

[12] Y. Shan and S. Kalyanaraman. Hybrid video downloading/streaming over peer-to-peer networks. In *IEEE Proceedings of the ICME*, 2003.

[13] A. Sharma, A. Bestavros, and I. Matta. dpam: A distributed prefetching protocol for scalable asynchronous multicast in p2p systems. Technical Report BUCS-TR-2004-026, 2004.

[14] T. F. Syeda-Mahmood and D. B. Ponceleon. Learning video browsing behavior and its application in the generation of video previews. In *Proceedings of the ACM Multimedia*, pages 119–128, 2001.

[15] Y. Wang, J. Ostermann, and Y.-Q. Zhang. *Video Processing and Communications.* Prentice Hall Publishing Company, 2002.

[16] X. Yang and G. de Veciana. *Service capacity of peer to peer networks.* In *IEEE Proceedings of the INFOCOM*, 2004.

[17] C. Zheng, G. Shen, and S. Li. Segment tree based control plane protocol for peer-to-peer on-demand streaming service discovery. In *Proc. of Visual Communication and Image Processing(VCIP)*, July 2005.