A Dynamic Caching Algorithm Based on Internal Popularity Distribution of Streaming Media

Jiang Yu[1,2], Chun Tung Chou[2]
[1]Dept. of Electronics and Information Engineering, Huazhong University of Science & Technology, China
[2]School of Computer Science and Engineering, University of New South Wales, Australia
frankyu@263.net, ctchou@cse.unsw.edu.au

UNSW-CSE-TR-0515
Sept 2005

Abstract

Most proxy caches for streaming videos do not cache the entire video but only a portion of it. This is partly due to the large size of video objects. Another reason is that the popularity of different part of a video can be different, e.g. the prefix is generally more popular. Therefore, the development of efficient cache mechanisms requires an understanding of the internal popularity characteristics of streaming videos. This paper has two major contributions. Firstly, we analyze two 6-month long traces of RTSP video requests recorded at different streaming video servers of an entertainment video-on-demand provider, and show that the traces provide evidence that the internal popularity of the majority of the most popular videos obeys a k-transformed Zipf-like distribution. Secondly, we propose a caching algorithm which exploits this empirical internal popularity distribution. We find that this algorithm has similar performance compare with fine-grained caching but requires significantly less state information.

1 Introduction

Streaming media has been widely used over the Internet in recent years. However, the growing use in streaming media, which generally has large size, can have a significant impact on the user perceived latency and network congestion. A popular approach to reduce the response time and backbone bandwidth consumption is to deploy proxy caches at the edge of the Internet. Due to the large size of streaming media, many caching algorithms for videos cache only a part of the video [1] [2] [3]. Most of these caching algorithms divide the video into segments and the caching unit is video segment rather than the entire video. Thus the popularity distribution of video segments has a direct impact on the design of efficient caching strategies. In order to make a caching decision, these schemes collect statistics on the access frequencies of the video segments. In other words, they use non-parametric statistics of video segment popularity. However, the choice of video segment granularity is not easy. With coarse granularity of segment, it cannot get the exact popularity difference between different parts of a video. With fine granularity of segment, it may take too much memory space to record the segment popularity information. In this paper, we design a new caching algorithm to solve this problem. This algorithm will estimate the segment popularity based on an empirical model for segment popularity distribution. Through using this model, the algorithm can avoid the trade-off discussed above and get good caching performance but requiring only to store a small amount of segment popularity information. We refer the popularity distribution of video segments of a video as the internal popularity distribution in this paper. Specifically, we address the following two questions in the following sections.

1. Can the internal popularity of streaming videos be described by some parametric statistical distribution?

2. If such statistical distribution can be found, how can we exploit that for caching?

Since the internal popularity of streaming videos depends on the user access behavior, a way to uncover the characters of the internal popularity distribution is through analyzing the traces of streaming media requests on the Internet. In this paper, we analyze two 6-month long traces of RTSP video requests recorded at two different streaming video servers of an entertainment video-on-demand provider. This work makes the following contributions: 1) We observed that the internal popularity of the majority of the most popular videos obeys a k-transformed Zipf-like distribution. Thus we can estimate the popularity of each segment using this empirical model which can be used for caching. 2) For k-transformed Zipf-like distribution, the segment popularity versus segment sequence number is a straight line in the logarithm of the transformed variables. This means that the popularity of all segments can be predicted by only knowing the state information of a few points on this straight line. Based on this observation, we propose a dynamic caching algorithm, called internal-popularity-based (IPB) caching algorithm, which maximizes the total bandwidth savings. The simulation results show that this algorithm performs well with significantly less state information.

The rest of this paper is organized as follows. Section 2 introduces the background on the server logs and presents the analysis that the internal popularity of the majority of the most popular videos in our logs obeys a k-transformed Zipf-like distribution. Section 3 presents the IPB caching algorithm. Section 4 reports the results of a performance study on the IPB caching algorithm. Section 5 reviews the related works. Finally, Section 6 concludes this paper and offers some future directions.

```
192.168.88.101-[09/Apr/2004:14:26:00+0800]
"GET content/dypk0000/zht00000/22jxj000.rm?
3200a45f49f07c08533c5eeea7065f27h5dcfe796h6h8d694h
ee894a3c700e791081499671286 1561 3:89:1:0:1:y331331
RTSP/1.0" 200 37153459
[Win98_4.9]_6.0.11.853_play32_RN10PD_en-US_586_axembed
[e7ca6a01-6b8e-11d8-fda7-79bf10a6f0c7]
[Stat3:13462|0|Resume|;832071|817421|STOP|;]
[Stat4:2audio/x-pn-realaudio|64_kbps_Stereo_Music_-_RA8
|7302|10|5|65965|117861|:video/x-pn-realvideo|N/A|26412|30
351930760 7972 824 74 0 10665
```

Figure 1: A complete record of a user access

2 Internal Popularity of the Video

2.1 Server log data

In this subsection, we will describe the media access logs that we used for our analysis. The traces were provided to us by www.cjmedia.com.cn. This website is a commercial entertainment website and it uses Helix Server as streaming media server. We use access logs from two different servers. The first server is called **Advanced Server (AServer)**. It hosted videos only for paid-up subscribers of the video-on-demand service, with videos in teleplays, popular movies and educational information. About 83.5% of these videos are longer than thirty minutes. The second server provided service to non-subscribers and will be referred to as **Free Server (FServer)**. In contrast to AServer, the videos in FServer mainly consists of advertisements and short clips. Out of these videos, 82.7% of them is less than fifteen minutes long and only 3.5% of them are longer than thirty minutes. All videos in both servers are encoded in real media [4] format. Clients can resume, pause, seek or stop during their viewing sessions. Here a session refers to a sequence of user requests corresponding to the same video. A typical example of a user access is shown in Figure 1. The information that is useful for our analysis in the record is the IP address of the client making the request, the time at which the request was made, the requested file name and the type of the request (e.g. resume, pause...). As shown in Figure 1, the IP address and the file name was recorded in the first line and the second line respectively. In order to protect the user anonymity, we use a private IP address instead of the original IP address here. The time stamp and the type of request were recorded in the line that begins with Stat3. Details on the packet format can be found in [5]. The traces record the user accesses from 01/04/2004 to 27/10/2004, totaling 210 days. The trace from FServer contains 521 videos and 37629 sessions. The trace from AServer contains 5688 videos and 54430 sessions.

2.2 Internal popularity analysis

In this subsection, we show that the internal popularity of the majority of the most popular videos from the two traces obeys a k-transformed Zipf-like distribution[1]. Considering the fact

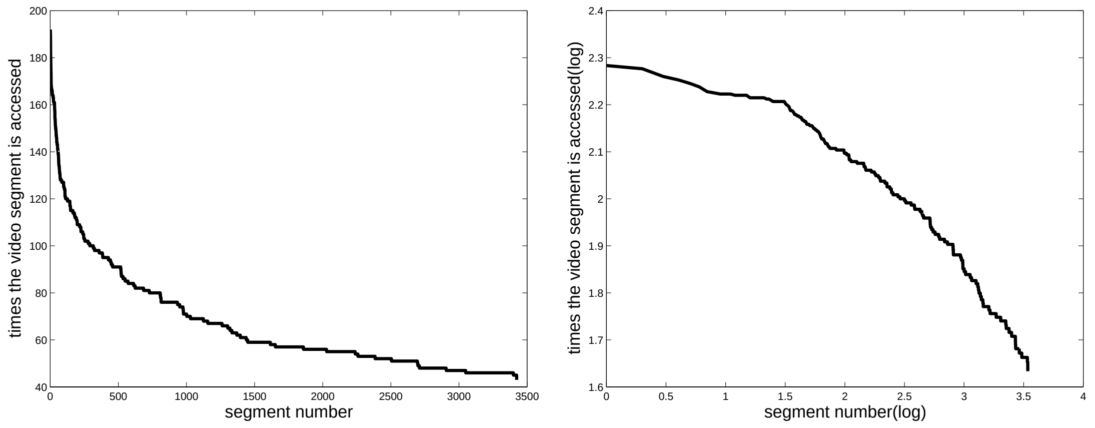
[1]Note that some trace analyses published in literature, such as [6] [7] [8], are based on the traces from the same website or company. The amount of traces in these papers ranges from one to five. In our paper, our video trace analysis is based on two traces from two different servers in the same website. These two servers have videos with different characters and provide service to different groups of clients respectively. Thus the

that there is not much point to analyze the less popular videos, we choose the videos which have been accessed at least 70 times in the 210 days. There are 121 videos in AServer and 29 videos in FServer which meet this requirement. The chosen videos make up only 2.1% and 5.6% of total videos in each server respectively. However they account for 22.2% and 31.3% of total number of sessions of each server. Thus, our analysis will focus on the internal popularity distribution of the 150 most popular videos.

For the purpose of analyzing the popularity of different parts of a video in our model, we divided the video into one-second segments because all the streaming media players use this precision. We define the segment popularity of a one-second video segment as the total number of times that segment has been accessed over the entire duration of the trace. We observe that the internal popularity of each video is an approximate decreasing function of video time because some users abort the session early—an observation that has been mentioned by many previous researches.

We first check whether the internal popularity of the chosen videos obeys Zipf-like distribution. If this hypothesis holds, a log-log plot of video segment sequence number versus segment popularity will be a straight line. An alternative method to check this hypothesis is to use linear regression on the logarithms of the segment sequence number and segment popularity. In linear regression, the closeness to a straight line fit can be measured by the coefficient of determination, denoted by R^2 [9]. The value of R^2 is between 0 and 1, where a value closer to 1 indicates a better fit.

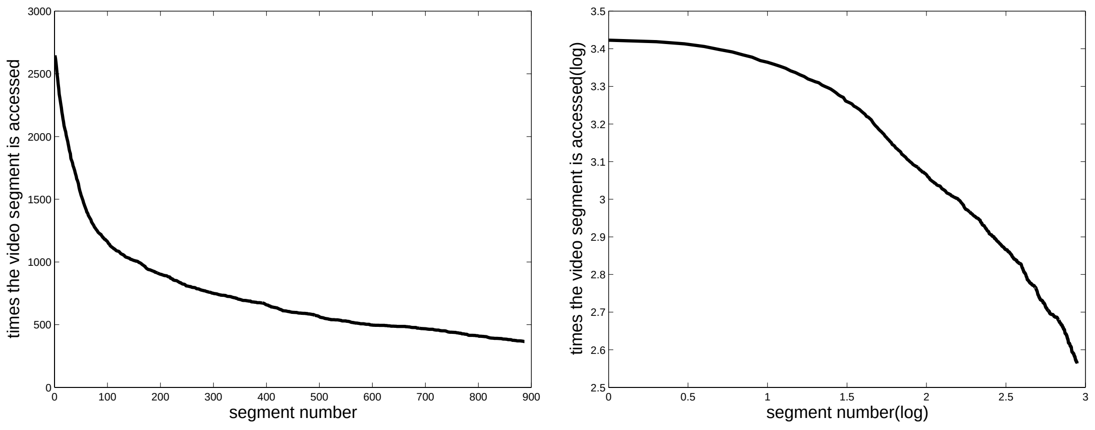
Figures 2 and 3 show the internal popularity of, respectively, the most popular videos in the two servers on both normal axes and logarithmic axes. It can be seen in these figures that the internal popularity of both videos does not seem to obey Zipf-like distribution. Instead of visually going through all the log-log plots for the 150 videos, we apply linear regression to the logarithmic data to check how well they fit a straight line. If we assume that a R^2 value greater than 0.90 means a good fit, then only 21.3% of the 150 videos has an internal popularity distribution which is Zipf-like, 20.7% in FServer and 21.5% in AServer.



(a) In normal axes (b) In log axes

Figure 2: Internal popularity for the most popular video in AServer

quality of information we have is comparable with those in previous work.



(a) In normal axes (b) In log axes

Figure 3: Internal popularity for the most popular video in FServer

An important contribution of our work is we propose to use the k-transformed Zipf-like distribution to model the internal popularity of the video. Originally, the k-transformed Zipf-like distribution is proposed in [10] to model long term file popularity. Here we use it to model the internal popularity distribution of streaming media. For a video, let x denote video segment sequence number and y denote the popularity of the segment. The k-transformation for the original x and y data is defined as follows:

$$x_k = \frac{x + k_x - 1}{k_x} \tag{1}$$

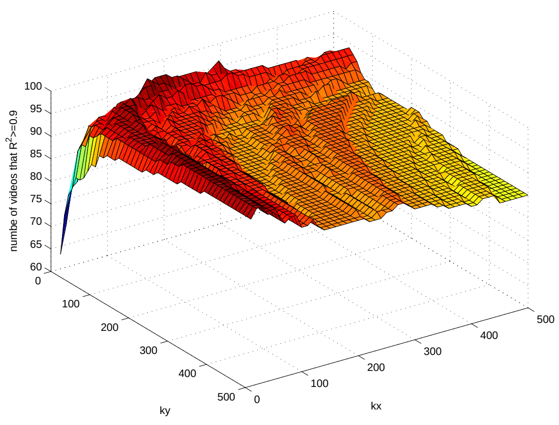
$$y_k = \frac{y + k_y - 1}{k_y} \tag{2}$$

where k_x and k_y are some positive constants. If the transformed data (x_k,y_k) obeys the Zipf-like distribution as described by Equation (3), then we say that the original data (x,y) obeys a k-transformed Zipf-like distribution.

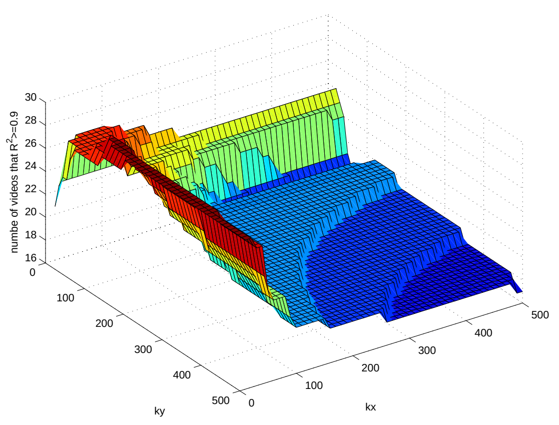
$$\log y_k = a \log x_k + b \tag{3}$$

Next, we will study how the internal popularity of popular videos obeys k-transformed Zipf-like distribution. Figure 4 shows the number of videos whose R^2 value is larger than 0.90 under different (k_x, k_y). It seems that we can get a better fit when k_x is about ten and k_y is large in our traces. Figure 5(a) and Figure 6(a) shows the typical relationship between the fitted straight line and original data on a log-log scale under different (k_x, k_y). The R^2 value of most videos will increase when k_y increases in our traces, as shown in Figure 5(b) where k_x=10. However there are also some exceptions, such as shown in Figure 6(b) where the R^2 value of these videos will decrease when k_y is larger than a threshold. It means that the value of (k_x, k_y) that gives the best fit can be different for different traces. Thus, we will suggest a scheme to adjust k_x and k_y dynamically in our caching algorithm in the following sections.

Table 1 shows the percentage of videos whose internal popularity obeys k-transformed Zipf-like distribution based on using the R^2 value. In FServer, all the popular videos have a R^2 value larger than 0.90 when k_x is 10 and k_y is larger than 200. And 80.2% of the videos in AServer have a R^2 value larger than 0.90 when k_x is 10 and k_y is larger than 400. Thus, we



(a) AServer



(b) FServer

Figure 4: Number of videos whose R^2 >=0.90 with different k_x's value and k_y's value

Table 1: Video distribution which R^2 >=0.90 for different k_y(k_x =10)

	k_y =50	k_y =100	k_y =150	k_y =200	k_y =250	k_y =300	k_y =350	k_y =400	k_y =450	k_y =500
AServer	68.6%	72.7%	76%	76.9%	77.7%	78.5%	79.3%	80.2%	80.2%	80.2%
FServer	89.7%	93.1%	96.6%	100%	100%	100%	100%	100%	100%	100%

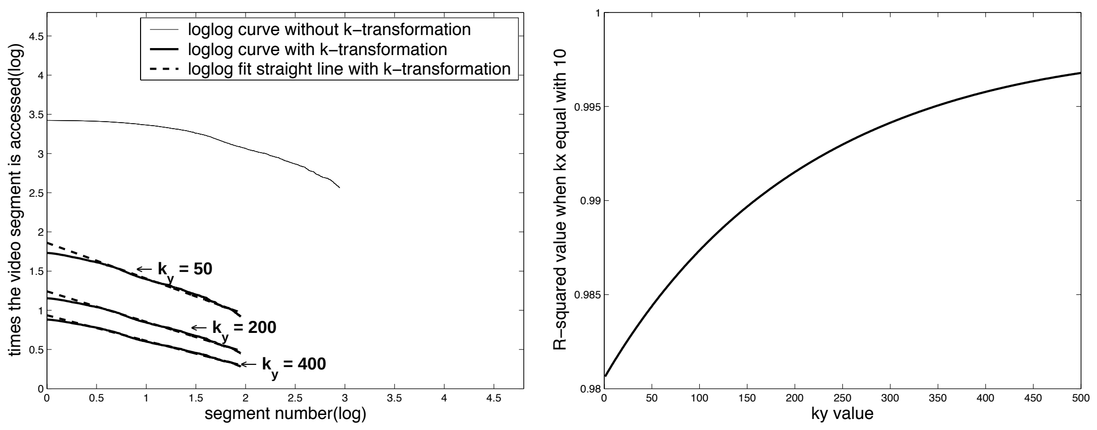
can conclude that the internal popularity of the majority of the most popular videos obeys a Zipf-like distribution after k-transformation. In the next section, we will design a dynamic cache scheme based on this observation.

3 Internal Popularity Based Caching Algorithm

Due to the large size of videos, most proxy caches only cache a portion of a video instead of the entire one. Therefore, a goal of the video caching algorithm is to determine the portion of the video to be cached in order to maximize the caching performance. We show in the Section 2 that the majority of the most popular videos has an internal popularity distribution which is Zipf-like after k-transformation. In this section we propose a new caching algorithm that we called internal popularity based (IPB) caching algorithm. This algorithm will estimate the internal popularity of each video based on k-transformed Zipf-like model and choose the appropriate segments to cache to minimize the bandwidth consumption of backbone network.

As the algorithm is based on the k-transformed Zipf-like model, the most important issue is to obtain accurate model parameters for each video; these include k_x and k_y in k-transformation as well as a and b in Equation (3)[2]. In our algorithm, we will use a common (k_x, k_y) for all videos. Initially, both k_x and k_y are set to one and we provide a dynamic update mechanism to update their values. For a and b, we can use linear regression to get their values for each video as the log-log plot of video segment sequence number versus segment popularity is a straight line after k-transformation. Considering the internal popularity of each video will be changed over time, we also provide a scheme to decide when to update the values of a and b. After getting the model parameter values for all videos, it is easy to estimate the

[2]Note that throughout this paper, a and b will be always used to refer to parameters in Equation (3).



(a) Fitted k-transformed Zipf-like distribution (k_x =10,k_y =50,200,400)

(b) R^2 value(k_x =10,k_y =1. . . 500)

Figure 5: Internal popularity for the most popular video in FServer

popularity of each segment of all videos based on Equations (1), (2) and (3). Then IPB will choose the most popular segments to cache subject to cache size constraint. We will describe the algorithm in details in the following subsections.

3.1 Updating k_x and k_y Dynamically

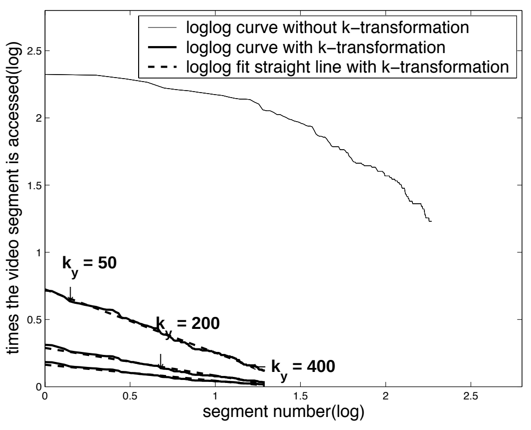
In this subsection, we will show how to update the values of k_x and k_y dynamically. In Section 2.2, it showed that the internal popularity of the majority of the most popular videos in our traces has a better fit with k-transformed Zipf-like distribution when k_x is small and k_y is large. However, we also found a few exceptions where the R^2 value decreases with increasing k_y. Thus, it may be difficult to find fixed values of k_x and k_y which are good for all proxy caches. In order to increase the robustness of the IPB caching algorithm, we propose a scheme to estimate the value of (k_x, k_y) dynamically.

Initially, both k_x and k_y are set to one. The algorithm will then update (k_x, k_y) when the number of RTSP requests is a multiple of a fixed update frequency F measured in terms of the number of RTSP requests. In order to choose the best (k_x, k_y) for the proxy cache, we defined a new metric called weighted average R^2 value (WAR) as follows:

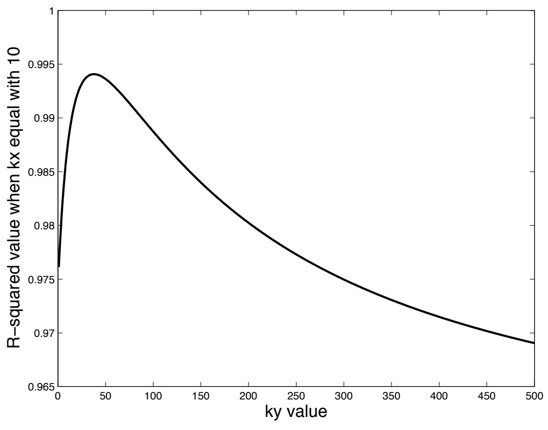
$$WAR = \frac{\sum_{i=1}^{n} w_i * R_i^2}{\sum_{i=1}^{n} w_i} \tag{4}$$

Here n denotes the number of videos and w_i is the weight for the R^2 value of video i which is chosen to be the popularity of video i.

In our algorithm, the (k_x, k_y) which achieves the largest WAR will be chosen in each update. That means we will prefer to choose the (k_x, k_y) that gives a better fit for the more popular videos. This is a nonlinear programming problem where k_x and k_y can be any positive real number. As nonlinear programming has been studied in many papers, many nonlinear programming algorithms can be used to solve this problem and we will not discuss it in this paper.



(a) Fitted k-transformed Zipf-like distribution (k_x =10,k_y =50,200,400)



(b) R^2 value(k_x =10,k_y =1...500)

Figure 6: Internal popularity for 10^{th} popular video in FServer

3.2 Recording and Updating User Access Information

The caching algorithm will need to store some user access information, i.e. video segment popularity information, in order to make caching decisions. However, before explaining how to record and update it, we will first introduce the concept of *record segment*. Since the internal popularity of the majority of the most popular streaming media obeys the k-transformed Zipf-like distribution, it means the log-log plot of video segment sequence number versus segment popularity will be a straight line. Thus we can estimate the popularity of different segments only based on few points on this straight line. In IPB caching algorithm, we will choose some segments from each video for this purpose and called these chosen segments as record segments in this paper. Let M denote the number of record segments and L denote the total number of segments in a video. The i-th record segment will be the j-th segment in the video, where j is the nearest integer less than or equal to j_r which can be calculated as follows:

$$j_r \quad = \quad 1 + (i-1) * \frac{L-1}{M-1} \tag{5}$$

The number of record segments in each video is the same regardless of the length of the video and its value can be any integer strictly larger than 2. This is because that we will get a perfect straight fit with R^2=1 if the number of record segments is only 2. Thus the WAR will always be 1 and the IPB caching algorithm cannot update k_x and k_y dynamically. Figure 7 shows an example with 7 record segments where each (segment number, segment popularity) pair recorded is shown as a dot.

Our algorithm will only record the popularity information of record segments instead of recording the popularity of all segments. This results in an efficient method to store internal popularity information. However, we cannot record the popularity information of record segments by only accumulating the total number of accesses to them because this will not allow us to track the temporal dynamics of internal popularity variation. We therefore introduce a window-based model in IPB caching algorithm for the purpose of forgetting the out-of-date information, as shown in Figure 8.

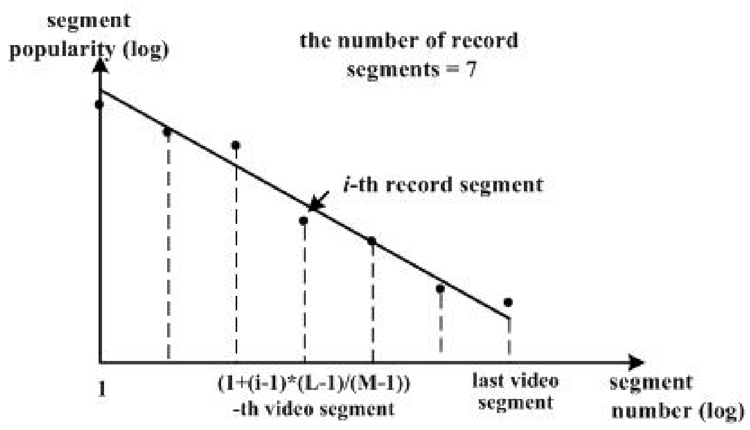


Figure 7: Record segments

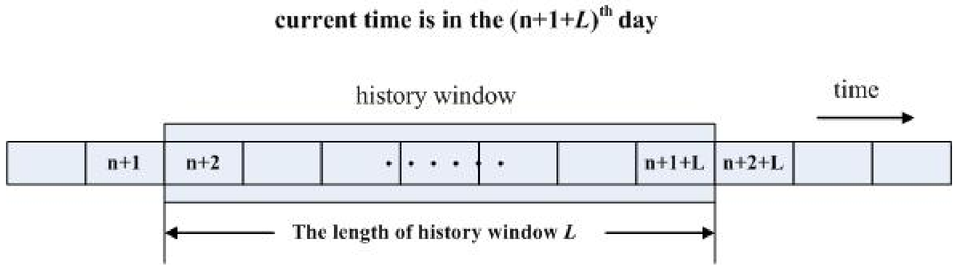

current time is in the (n+1+L)th day
history window
time
n+1
n+2
n+1+L
n+2+L
The length of history window L

Figure 8: History window model

Considering that most of videos are entertainment videos in our trace, we will use a day as the base unit of the length of history window in our algorithm. The popularity of record segments is stored on a per-day basis so that the out-of-date information can be removed later. In addition to the popularity of record segments, some other auxiliary information including the length of the video, the length of the cached portion of each video and the value of k-transformed Zipf-like distribution parameters a and b etc. will also be recorded.

The algorithm will update the recorded information in two ways. One is updating the information of the requested video when the proxy cache receives a new request from the user. We called it as "triggered update". It simply increases the popularity of the requested record segment in the current day. The second way is called "periodic update". It only happens when the history window moves. In this case, the algorithm will delete the the obsolete information from all videos whose information has been recorded. Especially, if the video's popularity become zero after updating, all the information about this video will be removed in order to make sure that only the information of videos that have been accessed in the history window will be recorded.

3.3 Updating the Value of a and b

If the recorded information has been changed, it may be necessary to re-calculate the internal popularity distribution of the video based on the new information. It was shown that the internal popularity of the video can be calculated based on Equations (1), (2) and (3) if we have the correct a and b parameters of the video. Thus, calculating the internal popularity of the video is essentially updating a and b based on the new recorded information.

Depending on whether triggered update or periodic update is used to update the recorded information, the IPB caching algorithm will update a and b in different ways. In triggered update, only the information of requested video, e.g. video i, will be updated. The algorithm will compare the optimal popularity threshold with the popularity of video i at first. The concept of optimal popularity threshold will be described in the Section 3.4. Here we just need to know that all the segments whose popularity is larger than or equal to this threshold will be cached. And we will use the popularity of the first record segment as video popularity because the first record segment is always the first segment of the video. If the popularity of video i is less than the optimal popularity threshold, the algorithm will do nothing because no segment of this video will be cached. Otherwise, the IPB algorithm will decide whether the model parameters of a and b of video i should be re-calculated. In order to make this decision, we should measure the goodness of fit between the existing model and the updated internal popularity information. We calculate the goodness of fit of video i, denoted by D_i, as follows:

$$D_i = \frac{\sum_{j=1}^{n}(\hat{y}_{ij} - \bar{y}_i)^2}{\sum_{j=1}^{n}(y_{ij} - \bar{y}_i)^2} \tag{6}$$

$$\hat{y}_{ij} = a_i * x_{ij} + b_i \tag{7}$$

$$\bar{y}_i = \frac{\sum_{j=1}^{n} y_{ij}}{n} \tag{8}$$

Here n is the number of record segments for video i. For video i, x_{ij} is the sequence number of the j-th record segment and y_{ij} is the popularity of the j-th record segment after receiving a new request. $\hat{y}_{ij}$ is the estimated popularity of the j-th record segment according to the model parameters a_i and b_i before receiving the new request. Expression D_i is very similar to R^2 except we use the new segment popularity information and old a and b values. If D_i value is larger than or equal to 0.90, it means that the existing a_i and b_i can model the internal popularity of video i well even the video has received a new request. Thus the algorithm does not need to do anything about updating the internal popularity distribution. Otherwise, the algorithm will re-calculate a_i and b_i for this video and record the new value of them.

Since triggered update occurs when a new RTSP request has arrived, only the a and b parameters for one video will be updated. When periodic update takes place, we will scan through all the videos to check whether their a and b values should be adjusted. The algorithm being applied to each video is the same as the triggered update case. The pseudocode is given in Figure 9.

3.4 Finding the Optimal Popularity Threshold

After obtaining the internal popularity distribution of each video, the algorithm will choose the portion of the video to cache to maximize the caching performance. In this subsection, we study the following optimal proxy caching problem. Let N denote the number of videos. The length of the i-th video is L_i segments (we assume here that the segment granularity of all videos are identical) and for the i-th video, the popularity of the x_{ij}-th segment ($x_{ij} = j = 1, ..., L_i$) is y_{ij}. We further assume that the internal popularity of all N videos obeys Zipf-like distribution after k-transformation and a common (k_x, k_y) will be used for all videos. For the i-th video, the model parameters (i.e. a and b used in Equation (3)) are denoted by a_i and b_i.

```
if update_mode==triggered_update
  result= video($i$).pop − threshold_pop;
  if result<=0
    return;
  else
    calculate $R^2$ based on $a$ and $b$ that recorded in the cache;
  end
  if $R^2$>=0.90
    return;
  else
    update $a$ and $b$ using linear regression
    based on new recorded access information;
  end
elseif update_mode == periodic_update
  for i=1:cached_video_num
    if(access information of video i has been changed)
      calculate $R^2$ based on $a$ and $b$ that recorded in the cache;
      if $R^2$ >=0.90
        return;
      else
        update $a$ and $b$ using linear regression
        based on new recorded access information;
      end
    end
  end
end
```

Figure 9: Calculating the internal popularity

With these assumptions, the relationship between x_{ij} and y_{ij}, $\forall i = 1, ..., N$ and $j = 1, ..., L_i$, can be described by

$$\log y_{k,ij} = a_i \log x_{k,ij} + b_i \tag{9}$$

$$x_{k,ij} = \frac{x_{ij} + k_x - 1}{k_x} \tag{10}$$

$$y_{k,ij} = \frac{y_{ij} + k_y - 1}{k_y} \tag{11}$$

The value of (k_x, k_y) can be estimated using the scheme described in Section 3.1. As explained in Section 3.3, the values of a_i and b_i for each video can be obtained by fitting a straight line to its record segments.

Given the internal popularity models of all the N videos, the optimal proxy caching problem is to determine the video segments to be cached so that bandwidth savings can be maximized subject to cache size constraint. For simplicity, we assume that each video segment uses b units of bandwidth and requires s units of storage space where the values of b and s apply to all video segments. Let C denote the size of the cache. Since the k-transformed Zipf-like model is a monotonically decreasing function, we can easily prove that only the prefixes of the videos will be cached. The optimal proxy caching problem can be formulated as follows:

Optimization problem (P1)

$$\max_{p_1,p_2,,,,,p_N} f(p_1,...,p_N) = \sum_{i=1}^{N}\sum_{j=1}^{p_i} y_{ij}b \tag{12}$$

$$\text{subject to } g(p_1,...,p_N) = \sum_{i=1}^{N} p_i s \leq C \tag{13}$$

$$p_i(i=1,...,N)\text{non-negative integers} \tag{14}$$

The decision variable p_i is the number of segments to be cached for the i-th video. The value of p_i is positive if some segments are cached, otherwise it is zero.

The above discrete optimization problem has been studied in the optimization literature [11] as well as in engineering e.g. rate-distortion trade-off in video coding [12]. Typical solution methods to this problem include Lagrange multiplier method [11] and dynamic programming [13]. We have chosen the former method here because it generally has lower complexity.

Let $\lambda > 0$ be the Lagrange multiplier. Consider the unconstrained optimization problem:
Optimization problem (P2)

$$\max_{p_1,p_2,,,,,p_N} f(p_1,...,p_N) - \lambda g(p_1,...,p_N) \tag{15}$$

For a given value of λ, let $p^*(\lambda)$ be the optimal N-tuple $(p_1, p_2,,,,,p_N)$ that maximizes **(P2)**, then it is proved in [11] that $p^*(\lambda)$ maximizes $f(p_1,...,p_N)$ subject to the constraint $g(p_1,...,p_N) \leq g(p^*(\lambda))$. Thus, if we can find a value of λ such that the corresponding $p^*(\lambda)$ has the property $g(p^*(\lambda)) = C$, then we know $p^*(\lambda)$ is the optimal solution to **(P1)**. However, there may not exist a $p^*(\lambda)$ such that the equality $g(p^*(\lambda)) = C$ holds, in this case we find the $p^*(\lambda)$ such that $g(p^*(\lambda))$ is as close to C as possible from below. Therefore, in order to solve **(P1)**, we must search for a suitable value of λ. It was shown in [14] that this can be done by a bisection search over λ.

Note that for a given value of λ, the problem **(P2)** can be decomposed into N independent optimization problems, one for each value of p_i $(i = 1,...,N)$:

$$\max_{p_i} \sum_{j=1}^{p_i} y_{ij}b - \lambda p_i s \tag{16}$$

It can show that the optimal solution to the above problem is given by the highest index j such that $y_{ij}b - \lambda s \geq 0$. Hence, the problem **(P2)** can easily be solved.

The above result means that segments whose popularity is larger than or equal to $\lambda s/b$ will be cached in order to get the maximal bandwidth saving in our algorithm. The quality $\lambda s/b$ will be referred to the *optimal popularity threshold*. We will use bisection search to find this value. As mentioned above, it is possible that $g(p^*(\lambda))$ is less than C. That implies there is some free cache space left. In order to get better performance, we will use a greedy algorithm to fill the remaining cache space up.

4 Performance Evaluation

4.1 Methodology

We evaluate the performance of the proposed IPB algorithm through trace driven simulations. The traces studied in section 2 will be used as the input data for our simulations. Considering

Table 2: Value of the Algorithmic Parameters

Name	Description	Value
a, b	parameters of k-transformed Zipf-like distribution as in Equation (3)	initial value = 0
k_x	parameter of k-transformation for segment sequence number	{1, 50*n (n=1,2,...,8)}
k_y	parameter of k-transformation for segment popularity	{1, 50*n (n=1,2,...,8)}
F	update frequency for k_x and k_y	100
M	the number of the record segments	4
C	the size of the cache space	10% of total size of all 150 videos

the proxy cache is placed at the edge of the network and provides service to different users, we combine the traces from the two different servers as the input data. Since only the most popular videos will be cached, we reduce the input trace to include only the 150 most popular videos in the combined trace. Four caching algorithms (IPB caching, Fine-grained caching, Exponential caching [15] and Zipf-like caching) will be implemented and the performance of them will be compared. In fine-grained caching, it will divide the video into one-second segments and the popularity information of all the segments will be recorded respectively. Thus it can distinguish the popularity difference in different segments exactly and is expected to have the best performance. We will use it as a benchmark to compare the performance with the other algorithms. For the Exponential caching, it uses an assumptive model of the internal popularity of the video. The video will be divided into the segment whose size increases exponentially and the assumptive model calculates the segment popularity by dividing the video popularity by the segment sequence number. The Zipf-like caching algorithm is similar to IPB caching. The only difference between them is that Zipf-like caching will assume that the internal popularity of popular videos obey Zipf-like distribution. We will study the gain in terms of performance by k-transformation through comparing the performance between Zipf-like caching and IPB caching algorithm.

The total bandwidth saving will be the performance metric. This metric is defined as the total units of bandwidth delivered from the proxy directly. A unit of bandwidth in our paper is equal to the number of bytes in a one-second segment of streaming video. For all algorithms, the history window model described in Section 3.2 will be used to control the useful information in the following simulations.

The standard value of IPB algorithmic parameters are list in Table 2. Unless otherwise specified, these standard values will be used in all simulations.

As it is difficult to find fixed values of (k_x, k_y) that are good for all caches, we initialize the (k_x, k_y) to (1 , 1) and propose a dynamic algorithm to estimate (k_x, k_y) in Section 3.1. In order to simplify the algorithm, we do not use continuous values for k_x and k_y but will choose them from the set {1, 50, 100, 150, 200, 250, 300, 350, 400}. The default number of the record segment is 4 and the standard value of update frequency for k_x and k_y is 100. The impact of these two IPB algorithmic parameters will also be studied in the following subsections.

4.2 Performance Comparison Between Different Caching Algorithms

In this subsection, we will study the performance of various caching algorithm with different length of history window. The length of history window will control the history of user access that is used to make a decision about the content of the cache. We choose the length of history window from 7, 15, 30, 60, 90 and 180 days. In other words, the length of history window will be one week, half month, one month, two months, three months and half year. With different lengths of the history window, the performance of the four algorithms is shown

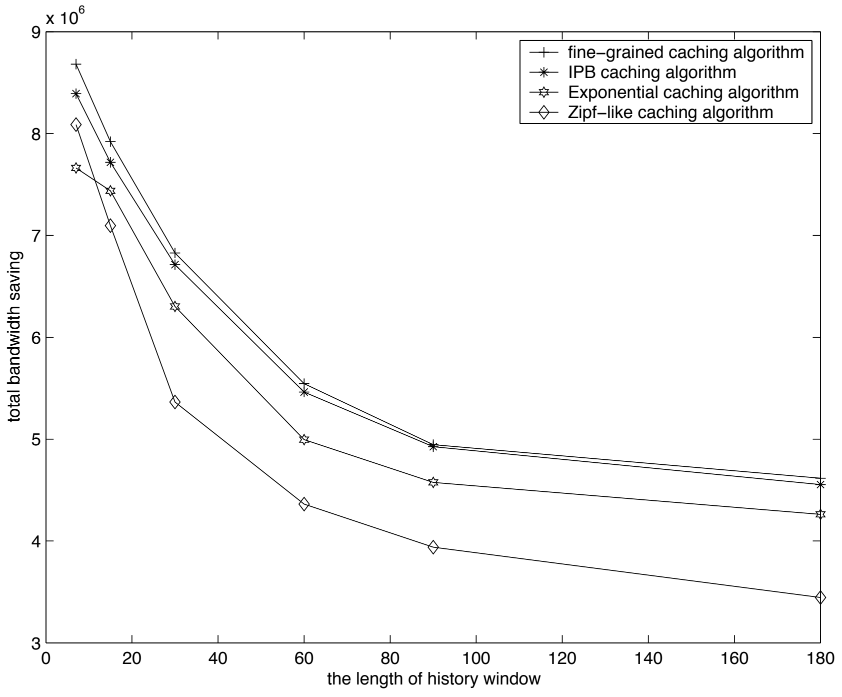


Figure 10: Performance comparison between different algorithms (record segment=4)

Table 3: optimal popularity threshold under different length of history window

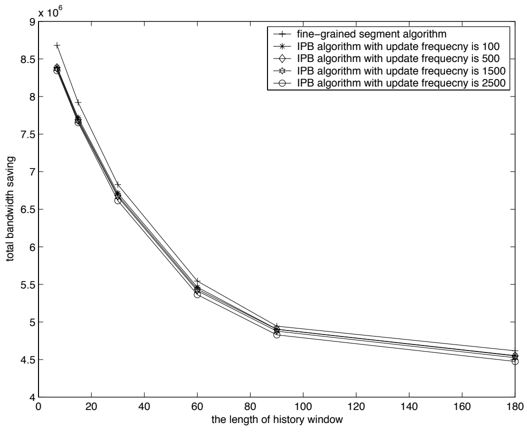
length of history window	7	15	30	60	90	180
optimal popularity threshold	1	2	8	23	34	60

as in Figure 10.

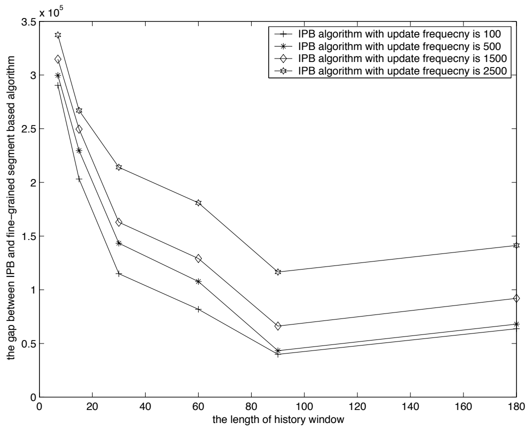
As depicted in Figure 10, the fine-grained caching algorithm has the best performance. It is obvious because the fine-grained caching algorithm will record the popularity information of all segments and can get the different popularity in different segments exactly. The IPB caching algorithm has very similar performance compared with the fine-grained algorithm though it only records the popularity information of four segments. The Exponential caching algorithm and Zipf-like caching algorithm will also record very few user access information but their performance is poorer than the fine-grained algorithm and IPB caching algorithm. That means k-transformed Zipf-like distribution model can describe the internal popularity of streaming video better than the Zipf-like distribution model and the assumed model in Exponential caching.

We also noted the performance will decrease when the length of history window increases. It is because the segment will accumulate more users' requests with longer history window and then the optimal popularity threshold for cached segment will be higher as shown in Table 3. However in the real trace that we used, not all videos are available at the beginning of the trace. Some of them are only introduced in the middle of the trace. Thus, the videos which are introduced later will spend a long time to accumulate its access frequency in order to be qualified to be cached because the optimal popularity threshold is very high.

From the Table 3, it seems a shorter history window gives the best performance in our trace. Since different severs have different timing for introducing new videos, e.g. some video servers update their content periodically [1] [16] [17], it may be difficult to argue what the



(a) performance comparison with different update frequency



(b) performance gap with different update frequency

Figure 11: Impact of update frequency of k_x and k_y

best length of the history window is. We will not discuss it in this paper since our purpose is only to demonstrate that IPB algorithm can have comparable performance with fine-grained segment algorithm for many values of history window length.

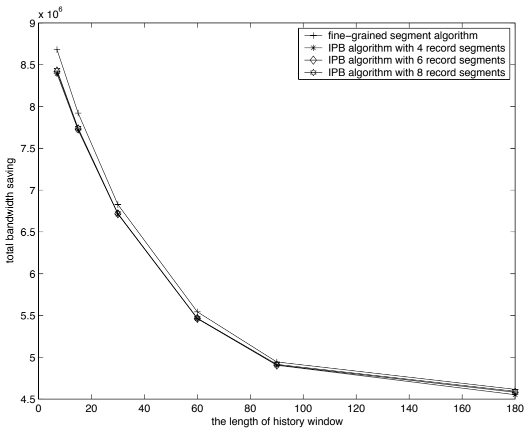
4.3 Impact of Algorithmic Parameters of IPB Caching Algorithm

Next, we will study the impact of algorithmic parameters on performance. We first conducted a set of simulations to investigate the impact of update frequency. In these simulations, the update frequency will be 100, 500, 1500 and 2500 respectively. We compare the performance between IPB and fine-grained caching algorithm as shown in Figure 11.

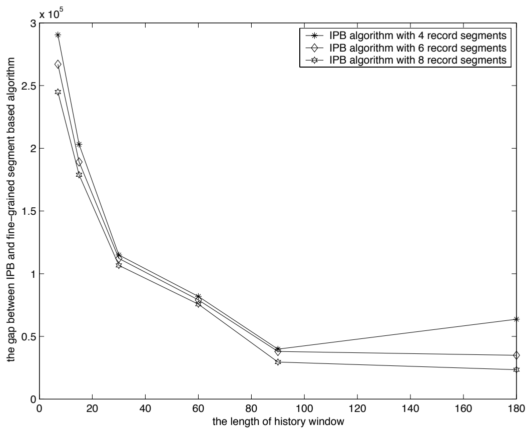
We observed from Figure 11(a) that the performance of IPB caching algorithm with different update frequency is very similar. We use the performance of fine-grained caching algorithm as a baseline and define the performance gap as the bandwidth saving of fine-grained algorithm minus that of IPB algorithm. As shown in Figure 11(b), the performance gap between IPB and fine-grained algorithm is small for small update frequency. Thus, frequent updates of k_x and k_y is helpful to improve the performance of IPB. However, this improvement is slight. For us, the most important advantage of updating the value of k_x and k_y dynamically is to avoid the difficulty of choosing suitable initial values of k_x and k_y; this also increases the robustness of algorithm.

We then conducted a second set of simulations to study the impact of different number of record segments. In the following simulations, the number of record segments will be 4, 6 and 8. In Figure 12, we show the performance of IPB with different number of record segments.

As shown in Figure 12(a), IPB caching algorithm with different number of record segments has similar performance with fine-grained caching algorithm. An increase in the number of record segments will decrease the performance gap between these two caching algorithms but the improvement is small for our data as shown in Figure 12(b). We explain this phenomenon as follows. We showed in Section 2 that the internal popularity of the majority of most popular videos obeys k-transformed Zipf-like distribution. We choose a video whose internal popularity fits the k-transformed Zipf-like distribution well and one that does not fit well



(a) Performance comparison with different number of record segments



(b) Performance gap with different number of record segments

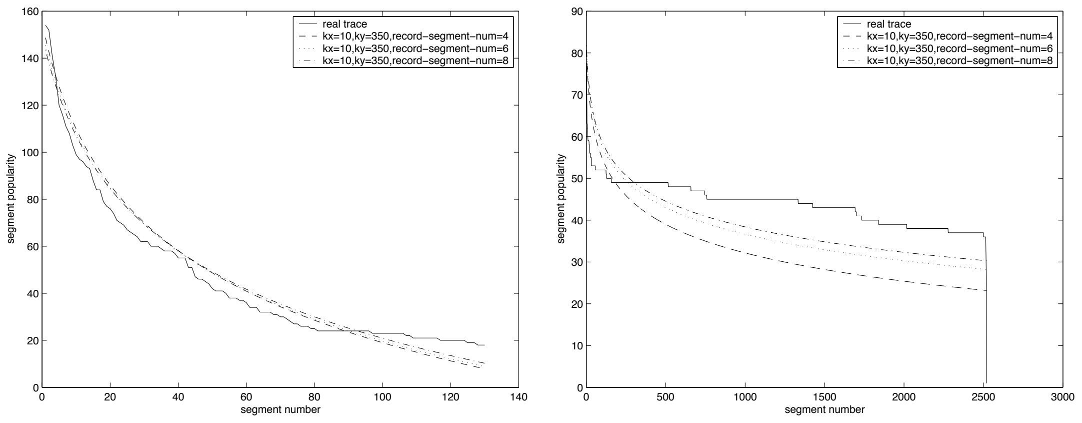
Figure 12: Impact of the number of record segments

to illustrate what happens when the number of record segments increases. We fix k_x as 10 and k_y as 350 which give a good fit between the internal popularity of the majority of most popular videos and the k-transformed Zipf-like distribution in our trace. Figure 13 shows the impact of the number of record segments on the prediction of internal popularity for these two chosen videos.

As shown in Figure 13(a), we can obtain a good estimation of segment popularity even if the number of record segments is four for the video whose internal popularity fits the k-transformed Zipf-like distribution well. The improvement in estimation is slight when the number of record segments increases. On the contrary, increasing the number of record segments can make a significant improvement in estimation of segment popularity of the video whose internal popularity does not fit k-transformed Zipf-like distribution well, as shown in Figure 13(b). Since the internal popularity of the majority of most popular streaming videos obeys k-transformed Zipf-like distribution in our traces, the impact of the number of record segments is slight in our simulations. However, increasing the number of record segments is still a good way to decrease the impact on performance by videos whose internal popularity distribution cannot be modeled well by k-transformed Zipf-like distribution.

4.4 Analysis on the Storage Space Required by the Recorded Information

We will analyze the amount of storage space required to record user access information by the four caching algorithms. For simplicity, we assume that the information such as the popularity of each video segment, length of the video etc. all require the same k units of storage space to record. In our simulations, fine-grained caching algorithm will record the information including the popularity of each segment in each day in the history window, the length of each video and the length of the cached portion of each video. Thus it will take $k * (total\ number\ of\ segments) * (the\ length\ of\ history\ window\ in\ days) + 2k * (total\ number\ of\ videos)$. In our trace, the total number of segment is 324510 and the total number of videos is 150. For the following calculation, we assume the length of history window is 7. The fine-grained caching algorithm will require at most $(2271870 * k)$ storage



(a) A video whose internal popularity obeys k-transformed Zipf-like distribution

(b) A video whose internal popularity does not obey k-transformed Zipf-like distribution

Figure 13: Impact of the number of record segments on the prediction of internal popularity

units for the 150 most popular videos. For Exponential caching, the following information will be recorded: the popularity of each video in each day in the history window, the length of each video and the length of the cached portion of each video. It will take at most $1350 * k$ storage units. For the IPB caching algorithm and Zipf-like caching algorithm, both of them will record the information including the popularity of record segment in each day in the history window, the length of each video, the length of the cached portion of each video and the value of parameters a and b. Thus, they will take at most $4800 * k$ storage units when the number of record segments is four.

According to this analysis, we find that fine-grained caching algorithm can get the best performance though consuming large storage space to record popularity information. All the other three caching algorithms which use a parametric model to estimate the segment popularity need only record few information. However only IPB caching algorithm has a very similar performance compared with the fine-grained caching algorithm in all situations. The performance difference between these two algorithms is at most 4% of total bandwidth saving but the state information required by IPB is less than 1% that of the fine-grained caching algorithm.

5 Related Work

There exists extensive research in the area of streaming media workload [8] [18] [19] [20] [21] because of its significance on proxy caching, multicast delivery and network management etc. However, relatively few of these studies are on modeling the internal popularity distribution.

Acharya et al [8] presented some basic properties on the internal popularity distribution of streaming media. They found that not all of the requests for streaming media completed the playback. It was found that only 55% of all requests played the entire video and the remaining 45% requests aborted early. If the popularity of a segment is defined as the total number of playback requests that last long enough to access the segment, the segment popularity distribution of streaming media is a decreasing function of playback length. Segments in the beginning of video are requested more often than later segments. However they did not continue to model the distribution of the internal popularity. W. Tang et al [10] analyzed the streaming media workload from enterprise servers. They propose an approximate model for the internal popularity where the section before the cut-off point is described by one sub-model and the section after that point is described by a different sub-model. The authors found that the location of the cut-off point can be different for different video. Thus this model may be difficult to be used for improving caching scheme. Our work is motivated by these studies. Our work provides a parametric statistical model to describe the internal popularity based on k-transformed Zipf-like distribution. Moreover, we show how to use this model as part of a caching algorithm.

Similar to most previous streaming media caching algorithms, our algorithm only caches the prefix of the video. Prefix caching was first proposed in [1]. In that paper, prefix caching is proposed to shield users from the delay, throughput and packet loss of the path from the server to the client in order to provide a smooth playback. Thus, the length of the video prefix only depends on the properties of the path between the server and the client. The difference in popularity of different videos does not play a role. In order to minimize backbone bandwidth consumption effectively, prefix caching has been combined with different transmission schemes with the consideration of different popularity in different videos in [17] [22] [23]. J. Almeida et al [22] focuses on combining a particular transmission scheme called bandwidth skimming with prefix caching and only study the proxy allocation in that case. A unified framework which can analyze various server scheduling, such as batching, patching and batch-patching, was built in [23] and a heuristic proxy caching algorithm was proposed in that paper. However, these papers only consider the video popularity but not the internal popularity of the videos.

To address this problem, segment based algorithms has attracted much attention in recent years. There are two types of segmentation strategies in previous research work. The first type is uniform segmentation [24] [25] [26]. For example, Rejaie et al proposed caching uniformly sized segments of layer-encoded video objects in [24]. In order to support partial viewing, Hofmann et al in [25] proposed SOCCER caching which grouped the uniformly sized segments into chunks and do prefix caching in each chunk. The second type is unequally-sized segmentation [2] [15]. For instance, media objects were divided into segments of exponentially increasing length and a specific number of segments were cached as prefix in [15]. As mentioned in Section 1, it is difficult to get a well-balanced trade-off between the veracity of segment popularity and the storage space used by recorded information for segment based algorithms. Our algorithm differs from the previous studies that we use an empirical model to estimate the segment popularity in order to avoid the difficulty in deciding the granularity of segmentation. Although the method of estimating the segment popularity has been studied before, few of them is based on an empirical internal popularity distribution model. The simplest method is to use the video or layer popularity as the segment popularity, such as in [24]. Obviously, this is not the best choice because different part of the video has different popularity. On the other extreme, the most exact method is to record the popularity information of all the segments directly, such as in [2]. However, it will take much memory space to record the user access information with fine granularity segmentation. In [15], the authors used an assumed formula to compute internal popularity distribution. They divided the video into segments of exponentially increasing length and calculate the segment popularity by the video popularity divided by segment sequence number. However, there is a lack of evidence that this method gives accurate prediction of segment popularity distribution. Through analyzing the two traces from two different servers, we provide some evidence that the internal popularity of majority of most popular videos obeys k-transformed Zipf-like distribution. Using this characteristic, our proposed algorithm will only record few user access information and can estimate all the segment popularity based on the k-transformed Zipf-like mode. We show in Section 4 that the algorithm gives good performance even if we record very few state information.

6 Conclusions and Future Work

The development of efficient caching algorithm for streaming media requires measuring, analyzing and parameterizing the internal popularity distribution of streaming media. In this paper, we analyzed two 6-month long traces of RTSP video requests recorded at two different streaming video servers of an entertainment video-on-demand provider. We study the relationship between the segment sequence number and segment popularity and implement a dynamic caching algorithm which we called IPB caching based on an internal popularity distribution that we observe. The analysis and simulation results show that 1) the internal popularity of the majority of the most popular streaming videos obeys a Zipf-like distribution after k-transformation. 2) The internal popularity distribution based caching algorithm performs well in different conditions with little user access information.

Our results are fundamentally based upon the workload that we captured and observed. It is clear that usage of streaming media in our environment is still relatively small. We will verify our observation with more traces in the future.

7 Acknowledgements

The authors would like to acknowledge www.cjmedia.com.cn for providing the trace data from their streaming media server, and the financial support from National Natural Science Foundation of China (No.60302004), the Australian Research Council (Grant LX0240468) and Natural Science Foundation of Hubei, China (No. 2005ABA264). This work was performed when the first author visits UNSW.

References

[1] S. Sen, J. Rexford, and D. Towsley, “Proxy prefix caching for multimedia streams,” in *Proc. IEEE INFOCOM99*, Mar. 1999.

[2] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, “Silo, rainbow, and caching token: Schemes for scalable fault tolerant streaming caching,” *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1328–1344, Sept. 2002.

[3] K. L. Wu, P. S. Yu, and J. L. Wolf, “Segmentation of multimedia streams for proxy caching,” *IEEE Transactions on multimedia*, no. 5, 2004.

[4] *Streaming Video with RealMedia*, http://www.mediacollege.com /video/streaming/formats/real-media.html.

[5] *Helix Universal Server Administrator Guide*, http://www.realnetworks.com/resources/documentation:R
Inc., 2004.

[6] L. Cherkasova and M. Gupta, “Analysis of enterprise media server workloads: access patterns, locality, content evolution, and rates of change,” *ACM/IEEE J.Transactions on Networking*, pp. 781–794, Oct. 2004.

[7] A. Mena and J. Heidemann, “An empirical study of real audio traffic,” in *Proc. IEEE INFOCOM00*, 2000.

[8] S. Acharya, B. Smith, and P. Parnes, “Characterizing user access to videos on the world wide web,” in *Proc. ACM/SPIE Multimedia Computing and Networking*, 2000.

[9] R. Jain, *The art of computer systems performance analysis.* Wiley, 1991.

[10] W. Tang, Y. Fu, L. Cherkasova, and A. Vahdat, “Long-term streaming media server workload analysis and modeling,” HP Laboratories Palo Alto, Tech. Rep. HPL-2003-23, Jan. 2003.

[11] H. EVERETT, “Generalized lagrange multiplier method for solving problems of optimum allocation of resources. operations research,” 1963, 11, 399-417.

[12] A. ORTEGO and K. RAMCHANDRAN, “Rate-distortion methods for image and video compression,” *Signal Processing Magazine, IEEE*, p. 23, 1998.

[13] D. P. BERTSEKAS, *Nonlinear programming.* Athena Scientific, 1999.

[14] Y. SHOHAM and A. GERSHO, “Efficient bit allocation for an arbitrary set of quantizers [speech coding],” *IEEE Transactions on Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing]*, p. 1445, 1998.

[15] K. Wu, P. Yu, and J. Wolf, “Segment-based proxy caching of multimedia streams,” in *Proc. 10th Int’l World Wide Web Conf. (WWW’01)*, May 2001.

[16] J. Liu, X. Chu, and J. Xu, “Proxy cache management of fine-grained scalable video streaming,” in *Proc. IEEE INFOCOM04*, 2004.

[17] B. Wang, S. Sen, M. Adler, and D. Towsley, “Optimal proxy cache allocation for efficient streaming media distribution,” in *Proc. IEEE INFOCOM02*, June 2002.

[18] J. M.Almeida, J. krueger, D. L.Eager, and M. K.Vernon, “Analysis of educational media server workloads,” in *Proc of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video. Port Jefferson, NY: ACM Press,*, June 2001.

[19] M. Chesire, A.Wolman, G. Voelker, and H. Levy, “Measurement and analysis of a streaming-media workload,” in *Proc. USENIX Symp. on Internet Technologies and Systems*, Mar. 2001.

[20] L. Cherkasova and M. Gupta., “Analysis of enterprise media server workloads: Access patterns, locality, dynamics, rate of change,” Internet Systems and Storage Laboratory, HP Laboratories Palo Alto, Tech. Rep. HPL-2002-56, Mar. 2001.

[21] T. Kuang and C. Williamson, “A measurement study of realmedia streaming traffic,” in *Proceedings of. ITCOM*, July 2002.

[22] J. Almeida, D. Eager, M. Ferris, and M.K.Vernon, “Provisioning content distribution networks for streaming media,” in *Proc. IEEE INFOCOM02*, 2002.

[23] C. Venkatramani, O. Verscheure, P. Frossard, and K. Lee, “Optimal proxy management for multimedia streaming in content distribution networks,” in *Proc. Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, May 2002.

[24] R. Rejaia, H. Yu, M. Handley, and D. Estrin, “Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet,” in *Proc. IEEE INFOCOM00*, Mar. 2000.

[25] M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang, “Caching techniques for streaming multimedia over the internet,” Bell Laboratories, Tech. Rep. BL011345-990409-04TM, Apr. 1999.

[26] S. Chen, B. Shen, S. Wee, and X. Zhang, “Adaptive and lazy segmentation based proxy caching for streaming media delivery,” in *Proc. NOSSDAV'03*, June 2003.